Culture Informatique
3 Heures Python/Arduino

Table des matières

1. Langages de programmation

1.1. Qu'est-ce qu'un langage de programmation ?

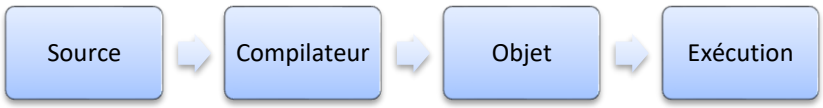
Un langage de programmation est un langage compréhensible par l'humain qui permet de décrire un algorithme par une succession d'instructions sera ensuite traduite en langage machine par un programme spécifique (le compilateur ou l'interpréteur).

1.2. Interprétation et compilation

Une grande différence entre les langages de programmation est la façon dont ils sont traduits en langage machine.

1.2.1. La compilation

Elle consiste à traduire entièrement un langage de haut niveau (s'approchant davantage du langage humain) en un langage machine (problèmes de portabilité). Le programme chargé de faire cette traduction s'appelle le compilateur.



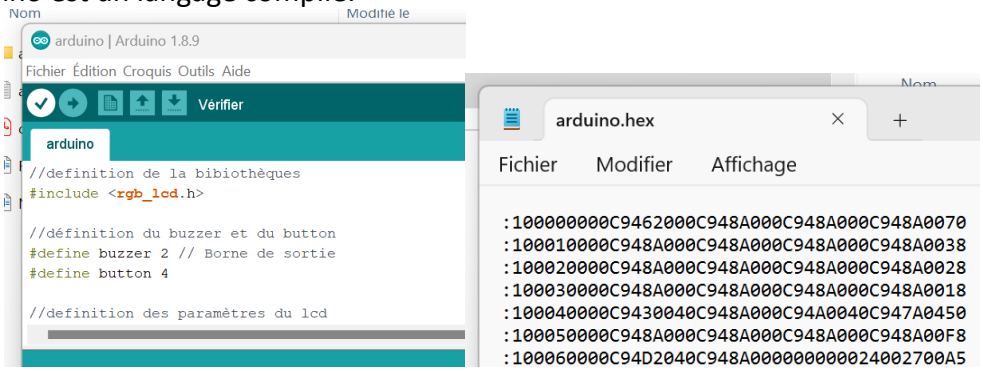
1.2.2. L'interprétation.

L'exécution est effectuée directement à partir du fichier source : l'interpréteur lit les instructions les unes après les autres, et envoie au fur et à mesure sa traduction au processeur.

Python est un langage interprété, semi-compilé. On peut ainsi bénéficier

- D'une utilisation en console,
- D'une détection dynamique des erreurs de syntaxe.
- L'exécution du programme commence par une compilation partielle, nécessitant une syntaxe complète.
- Le code généré est exécuté par une machine Python préalablement installée sur la machine

Le langage Arduino est un langage compilé.



2. Structuration d'un programme

2.1. Notion de programme

Un programme est constitué d'une succession d'instructions.
Les délimitations des blocs d'instructions se font par

Python	Arduino
Indentation 4 espaces Le caractère **:**	**;** Les caractère { }

Les instructions peuvent être :

- une affectation : L'état mémoire est modifié par l'évaluation d'un expression
- l'utilisation d'une fonction, ou d'une méthode sur un objet, ou un calcul
- une structure composée (conditionnelle, itérative, ...)

Des commentaires destinés à l'humain peuvent être placés pour aider à la compréhension de l'algorithme.

Python	Arduino

# commentaires sur 1 ligne	`// Commentaire 1 ligne` 	
	`/* commentaire multi lignes`
	`commentaire multi lignes`
	`*/`

2.2. Notion de « variable »

Une variable est un identifiant de l'adresse mémoire associée et possède un type (la nature de l'objet stocké dans la variable).
En informatique un variable peut être une **constante** !

Python	Arduino
GRAVITATION = 9.81 **# MAJUSCULE**	#define colorR 23;

En Python, les variables bénéficient d'un typage dynamique : le type est détecté automatiquement lors de la création de la variable par affectation. Il n'est donc pas nécessaire de déclarer la variable avant de l'utiliser (comme dans la plupart des autres langages).

Python	Arduino
vitesse = 20	int vitesse ; vitesse = 20 ;

Remarque : Même si en Python on parle de « variable » ce sont déjà des objets.

2.3. Notion d'objet

Un objet est une instance d'une classe. C'est une variable d'un type « élaboré ». Il dispose de propriétés et de méthodes.
La notation pointée permet de manipuler les propriétés et méthodes :

- `objet.propriete`
- `objet.methode(paramètres)`

2.4. Affectation

L'affectation est l'action consistant à donner une valeur à une variable (var ← (valeur ou référence)

```
a = 2 # affectation
```

Une affectation peut prendre en argument le résultat de l'évaluation d'un expression :

```
a = a + 4**a
```

2.5. Les fonctions

Une fonction est un morceau de programme que l'on isole. On définit sa signature en indiquant ses paramètres et leurs types ainsi que la nature de ce qu'elle renvoie.

Python	Arduino

`def nom_de_la_fonction(x, y):`
`  """Description"""`
`  instructions`
`  return resultat`
	`void setup() {instructions}`
	
	`double calculerPGDC(int nombre1, int nombre2)`
	`{`
	`instructions`
	`return nombre3;`
	`}`

Arduino distingue les procédures (qui ne renvoient rien) des fonctions.
En l'absence du mot clé `return`, Python renvoie `None`.

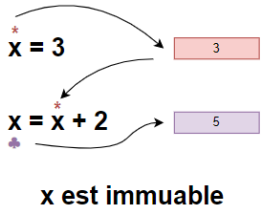
On utilise la fonction par un appel de la forme
`nom_de_la_fonction(arguments)`

Si la fonction renvoie un résultat, celui-ci devra être mémorisé dans une variable, l'instruction sera de la forme :
`var = nom_de_la_fonction(arguments)`

Les arguments d'une fonction sont les valeurs réelles passées à la fonction. Selon les situations on parle de passage par valeur ou passage par référence.

2.6. Muable ou immuable, valeur ou référence

Plusieurs types d'objets python (booléens, entiers, flottants, chaînes et tuples) sont immuables. Cela signifie qu'après avoir créé l'objet et lui avoir attribué une valeur, vous ne pouvez pas modifier cette valeur.



x est immuable

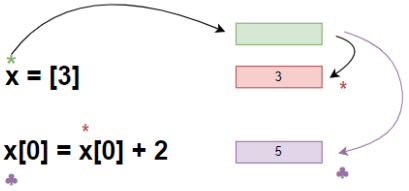
Les objets de type immuable ne peuvent pas être modifiés une fois créés, pourtant ce code ne provoque pas d'erreur :

```
>>> nb_un = 32
>>> nb_un = 40
```

En réalité, même si le nom est resté le même, ce n'est plus le même objet !

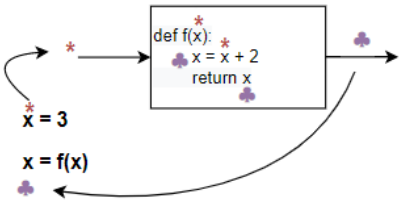
Contrairement aux types intégrés tels que int, float, bool, string, unicode, tuple qui sont des types d'objets immuables, les types list , dict , set sont des types d'objets muables. Les classes personnalisées sont aussi généralement modifiables.

```
>>> x = ["red", "blue", "green"]
>>> id(x)
87074096
>>> x[0] = "pink"
>>> id(x)
87074096
```

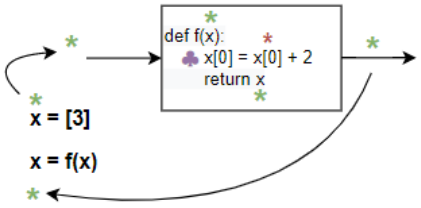


x est muable

Cette notion prend toute son importance lors du passage d'argument à une fonction.



On passe la référence
On reçoit la référence équivalent
passage par valeur



On passe la référence
On reçoit la même référence
Modification "en place"
return INUTILE

2.7. *La structure conditionnelle*

Elle permet de modifier la séquence réalisée à la suite d'un test booléen (Condition vraie)

```
# Si condition alors bloc1 sinon bloc2
```

Python	Arduino

`if a > 5:`
`  a = a + 1`
`else:`
`  a = a - 1`
	`if (digitalRead(pinCapteur) == HIGH) {`
	`  // si un train est detecte`
	`  digitalWrite(pinDEL, HIGH);  // allume la DEL`
	`  }`
	`  else { // sinon`
	`  digitalWrite(pinDEL, LOW);  // eteint la DEL`
	`  }`

Sinon (`else`) et le bloc 2 sont optionnels. En Python Lorsque le bloc2 contient lui aussi une structure conditionnelle, on peut écrire `elif`

Python	Arduino

`if a > 5:`
`  a = a + 1`
`else:`
`  if a < 3:`
`  a = a – 1`
`else:`
`  a = 0`
	`if (a > 5) {`
	`  a = a + 1 ;`
	`}else{`
	`  if (a < 3) {`
	`  a = a – 1 ;`
	`}else{`
	`  a = 0 ;`
	`}`

2.8. Les structures itératives

Il faut distinguer deux types de structures itératives ; celles où le nombre d'itérations est connu dès le début (itération sur un ensemble fixe de valeurs), et celles où l'arrêt de l'itération est déterminé par un test.

2.8.1. Boucle bornée (à privilégier)

La structure itérative `for` permet d'itérer sur un nombre fini de valeurs. Elle s'utilise avec un objet itérable quel qu'il soit.

 # pour indice variant de valeur de début à valeur de fin par pas défini

Python	Arduino

`for i in range(10, 20, 5):`
`    blocInstructions`
	`for (int compteur=20 ; compteur>=0 ; compteur--)`
	`// ici on compte à rebours, en décrémentant le compteur`
	`{`
	`Serial.println(compteur);`
	`}`

La variable peut être évaluée dans la boucle mais ne doit pas être modifiée !

 # pour chaque élément d'un objet itérable

Python	Arduino
`for element in objet_iterable:` `    blocInstructions`	**INEXISTANT**

2.8.2. Boucle non bornée

On itère la boucle tant qu'une certaine condition est vérifiée (vraie)

 # Faire action pendant que condition

Python	Arduino

INEXISTANT	`do {`
	`   // instruction(s);`
	`}while (condition);`

 # Tant que condition faire action

Python	Arduino

`while cond:`
`    instructions`
	`while (condition) {`
	`   // instruction(s);`
	`}`

3. Objets et méthodes

3.1. Les nombres

3.1.1. Python

Trois classes principales (entiers, flottants, complexes).

Les complexes s'écrivent sous la forme a + bj. Le réel b doit être spécifié, même si b = 1, et accolé à la lettre j.

Par exemple 1+1j désigne le complexe 1 + i.

Attention un nombre flottant est une représentation approximative d'un nombre réel :

```
>>> 0.1+0.1+0.1 == 0.3
False
```

3.1.2. Arduino

Arduino dispose de type primitif int, long, short , byte, double, float, char

3.2. Les booléens et les tests

Les deux éléments de la classe des booléens sont `True` et `False` (avec les majuscules).

3.3. Les chaînes de caractères

Il s'agit d'un assemblage de caractères. Une chaîne est un objet.
https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/
L'accès à un caractère d'une chaîne se fait par indexation

3.4. Les tuples

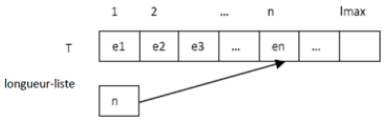
Les `tuples` sont des n-uplets. Ils sont notés entre parenthèses :

```
(1,2,3,4) # enumeration des objets entre ()
```

3.5. Les listes

Une liste est une suite ordonnée d'objets, de même type

- Représentation contiguë



- Représentation chainée

- Python est un langage de haut-niveau.
- Le type list essaie de prendre les avantages des deux structures précédentes.
- Le type list va plus loin : les éléments peuvent être de types différents.

On accède aux éléments d'une liste par leur index.
Les éléments d'une liste sont modifiables.
https://www.arduino.cc/reference/en/language/variables/data-types/array/

3.6. Les dictionnaires

Un dictionnaire (classe `dict`) permet de définir une association entre un ensemble d'indices (appelés clés, et pouvant être de type quelconque) et des valeurs. L'association entre une clé et une valeur est appelée un objet ou une entrée du dictionnaire (item)

4. Modules complémentaires Python

4.1.1. Présent ou pas

Python dispose d'un certain nombre de fonctions intégrées qui sont automatiquement chargées au démarrage et sont toujours disponibles, telles que print() et input() pour les E/S, les fonctions de conversion de nombres int() , float() , complex() , les conversions de type de données list() , tuple () , set() , etc.

En plus des fonctions intégrées, un grand nombre de fonctions prédéfinies sont également disponibles dans le cadre de bibliothèques fournies avec les distributions Python. Ces fonctions sont définies dans des modules appelés modules intégrés.

Les modules intégrés sont écrits en C et intégrés au shell Python. Chaque module intégré contient des ressources pour certaines fonctionnalités spécifiques au système telles que la gestion du système d'exploitation, les E/S de disque, etc. La bibliothèque standard contient également de nombreux scripts Python (avec l'extension .py) contenant des utilitaires utiles.

Pour afficher une liste de tous les modules disponibles sur l'environnement utilisez la commande suivante dans la console Python :

```
>>> help('modules')
```

Avant d'utiliser un module spécifique il faudra l'installer dans l'environnement !

4.1.2. Importation du module

De nombreuses fonctions sont définies dans des modules spécialisés qu'il faut importer dans le programme avant de pouvoir les utiliser.

- Import simple du module (`#include <SPI.h> pour arduino`)

```
import mod
```

- Import avec alias

```
import mod as alias
```

L'utilisation de préfixes permet d'utiliser des fonctions ayant éventuellement même nom et se situant dans des modules différents.

- Import d'une fonction d'un module

```
from mod import fonct
```

- Import de toutes les fonctions d'un module

```
from mod import *
```

Cette technique doit être évitée car elle charge l'espace de nommage et pose question lors du chargement de plusieurs modules

Une fois importé, Utiliser help() pour avoir la liste de toutes les fonctions du module.

4.2. Le module math

Ce module contient les fonctions et constantes mathématiques usuelles.

4.3. Le module numpy (calcul numérique)

Ce module définit un certain nombre d'objets indispensables en calcul numérique, en particulier l'objet matriciel, et toutes les règles associées. Souvent importé sous l'alias `np`.
Le module `numpy` contient lui-même des sous-modules, Si on n'utilise qu'un sous-module particulier, on peut importer uniquement ce sous-module :

```
import numpy.linalg as al
```

Quelques différences entre les listes et les `np.array` :

- Homogénéité: toutes les entrées doivent être de même type
- Le format est immuable. La taille est définie à partie de la première affectation. Pour initialiser, il faut donc souvent créer un tableau préalablement rempli de 0 ou de 1.

Les `np.array` permettent le calcul matriciel ce qui n'est pas le cas des `list`

4.4. Le module scipy (calcul scientifique)

Le module `scipy` regroupe un certain nombre de fonctions de calcul scientifique (algorithmes classiques de calcul numérique).

4.5. Le module matplotlib (tracé de courbes)

Ce module donne un certain nombre d'outils graphiques, notamment pour le tracé de courbes. On utilise le plus souvent le sous-module `matplotlib.pyplot`, souvent importé sous l'alias `plt`.

4.6. Autres modules (random, time, pandas ,…)

Le site https://pypi.org/ recense plus de 450 000 paquets Python … Heureusement, Les fonctionnalités considérées comme indispensables ont été ajoutées par défaut à Python. Elles ne nécessitent aucune installation supplémentaires.

4.7. Lecture et écriture de fichiers

On peut utiliser les fonctions standard et importer le module csv ou pandas qui facilitent l'exploitation des données.
On collecte des caractères qu'il faudra toujours transformer en nombre pour faire des opérations.
On doit toujours

- Designer le chemin du fichier
- Le mode d'ouverture

```
# ' r'  : ouverture en lecture seule
# ' w'  : ouverture en écriture (efface le contenu précèdent)
# ' a'  : ouverture en ajout (écrit à la suite du contenu précédent)
```

On peut ensuite

- Lire ou écrire une ou plusieurs lignes

Exemple

```
import csv
def import_csv (cible):
    '''
    cible : chemin absolu du fichier"
    rep : liste [ []...]
    '''
    rep = []
    with open(cible, 'r',encoding='utf-8', newline='') as fichier:
        fichier.readline()
        reader = csv.reader(fichier, delimiter=';')
        for row in reader:
            rep.append(row)
    return rep
```

Remarquer :

`encoding='utf-8'`

`delimiter=';'`

`#row type list`

5. Exercices commentés

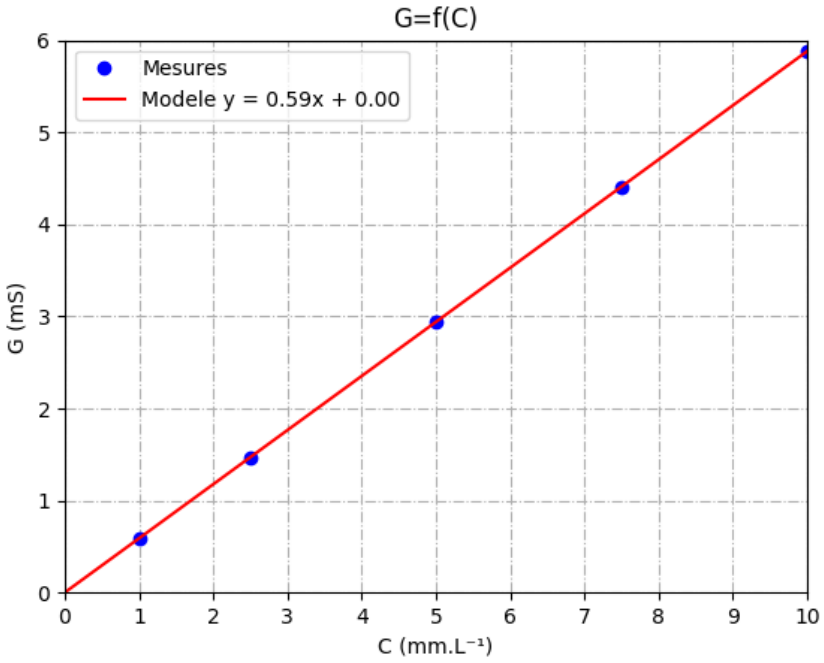
5.1. Chap 2 Méthodes physiques d'analyse

p45ex21.py

```
import matplotlib.pyplot as plt
import numpy as np
# Données expérimentales
x = [10.0, 7.5, 5.0, 2.5, 1.0] # C en mm.L⁻¹
y = [5.88, 4.41, 2.94, 1.47, 0.59] # G en mS
# Affichage
plt.title("G=f(C)")
plt.xlabel("C (mm.L\u207B\u00B9)") # unicode ⁻¹
plt.ylabel("G (mS)")
plt.plot(x, y, "bo", label='Mesures') # symbole bl
plt.legend()
plt.axis(xmin=0, xmax=10, ymin=0, ymax=6)
plt.grid(linestyle="-.")
plt.xticks(range(11)) # nb de lignes
# Modélisation pôlynome ordre 1
coef = np.polyfit(x, y, 1) # renvoie array([0.5878424 , 0.00121951])
# Calcul du modèle pour 5 valeurs prises entre 0 et xmax
xmod = np.linspace(0, max(x), 5)
modele = [coef[1] + coef[0] * val for val in xmod]  # Calcul du modèle
plt.plot(xmod, modele, color = "red", label =f"Modele y = {coef[0]:.2f}x + {coef[1]:.2f}") # "r-"
plt.legend()
plt.show() # affiche la figure à l'écran
```

```
import matplotlib.pyplot as plt
import numpy as np
# Données expérimentales
x =
y =
# Affichage
plt.title
plt.xlabel
plt.ylabel
plt.plot(x,y,"bo",label="points\
expérimentaux")
plt.axis(xmin=0,xmax=10,ymin=0,ymax=6)
plt.grid(linestyle="-.")
plt.xticks(range(11))
# Modélisation polynôme ordre 1
modele=np.polyfit(x,y,1)
```

p45ex21.py

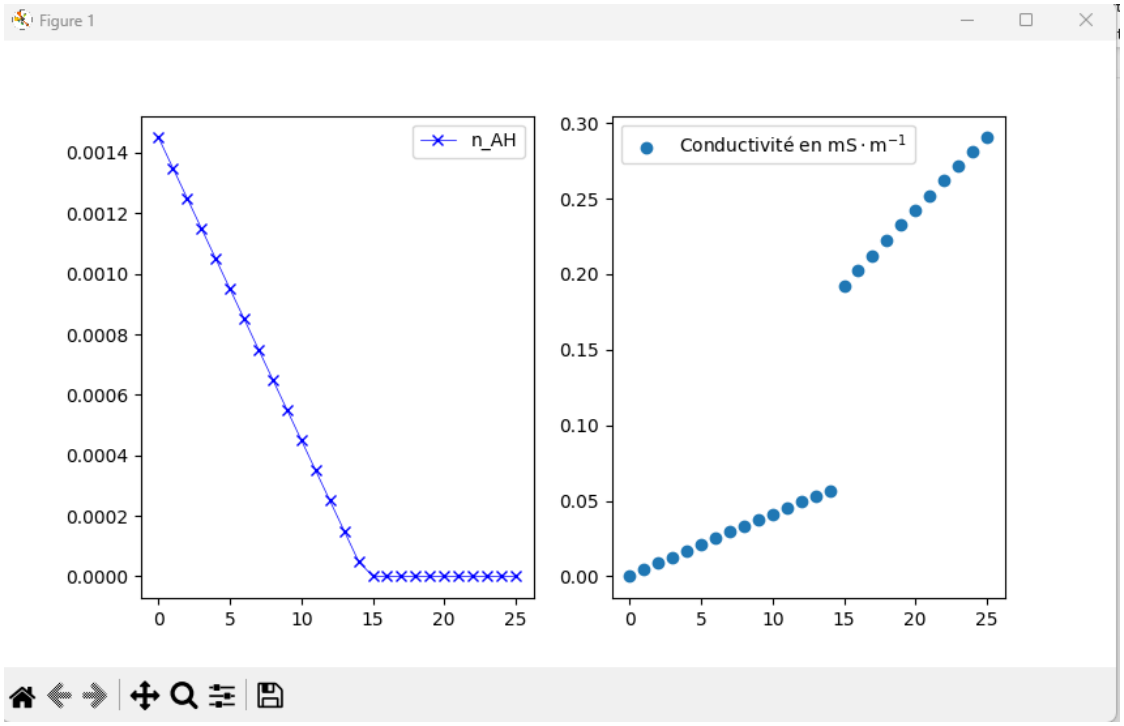


p48act3.py

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

def calcul_quantites_avant_et_a_equivalence(i_ajout):
    v.append(i_ajout) # en nombre d'ajout
    n_AH.append((0.145*10.0 - 0.100*(VDOSE*i_ajout)) * 0.001) #
    n_HO.append(0)
    n_Na.append(0.10 * i_ajout * 0.001)
    n_A.append(0.10 * i_ajout * 0.001)

def calcul_quantites_apres_equivalence(i_ajout) :
    # pass
    v.append(i_ajout)
    n_AH.append(0)
    n_HO.append(0.100*(VDOSE*i_ajout)* 1E-3)
    n_Na.append(0.100*(VDOSE*i_ajout)* 1E-3)
    n_A.append(n_A[-1])

VEQUIVALENCE = 14.5  # Volume équivalence en mL
VEAU = 200 # Volume d'eau ajouté en mL
VDOSE = 1 # Dose délivrée à chaque ajout du réactif titrant en mL
n_AH, n_HO, n_Na, n_A, v = [], [], [], [], [] # initialisation de 5 listes de valeurs
for ajout in range(0, 26, 1) : # nb ajout volume titrant
    if ajout*VDOSE <= VEQUIVALENCE :
        #appelle une fonction qui calcule les quantités avant et à l'équivalence
        calcul_quantites_avant_et_a_equivalence(ajout)
    else :
        #appelle une fonction qui calcule les quantités après l'équivalence
        calcul_quantites_apres_equivalence(ajout)
# Expression de la conductivité
# Eau + V titrant + titree 10mL acide aqueux
conductance = (20*np.array(n_HO) + 5.0*np.array(n_Na) + 4.1*np.array(n_A)) / (VEAU*1E-3 + np.array(v)*1E-3 + 10E-3)

# affichage des courbes
plt.subplot(121) # subplot permet d'afficher deux graphes
plt.plot(v, n_AH,'bx-', linewidth=0.5, label="n_AH")
plt.legend()
plt.subplot(122)
plt.plot(v, conductance, 'bo', label=r"Conductivité en $\mathrm{mS\cdot m^{-1}}$") # Affichage en Latex
plt.legend()
plt.show()
```

```python
import matplotlib.pyplot as plt
import numpy as np
…
VE = 14.5  #saisie du volume équivalent en mL
Veau = 200 #saisie du volume d'eau ajouté en mL
n_AH, n_HO, n_Na, n_A, V = [], [], [], [], [] # initialisation de 5 listes
for i in range(0,26,1) :
    if i <= VE :
        #appelle une fonction qui calcule les quantités avant et à l'équivalence
        calcul_quantites_avant_et_a_equivalence(i)
    else :
        #appelle une fonction qui calcule les quantités après l'équivalence
        calcul_quantites_apres_equivalence(i)
#Expression de la conductivité
Conduct = (20*np.array(n_HO)+5.0*np.array(n_Na)+4.1*np.array(n_A)) \
/(Veau*1E-3+1E-3*np.array(V))
#affichage des courbes
plt.subplot(121) #subplot permet d'afficher deux graphes
plt.plot(V,n_AH,'bx-',linewidth=0.5,label="n_AH")

…
plt.legend()
plt.subplot(122)
plt.scatter(V, Conduct, label="Conductivité en $\mathrm{mS\cdot m^{-1}}$")
plt.legend()
plt.show()
```

p48act3.py



5.2. Chap 3 méthodes chimiques d'analyse

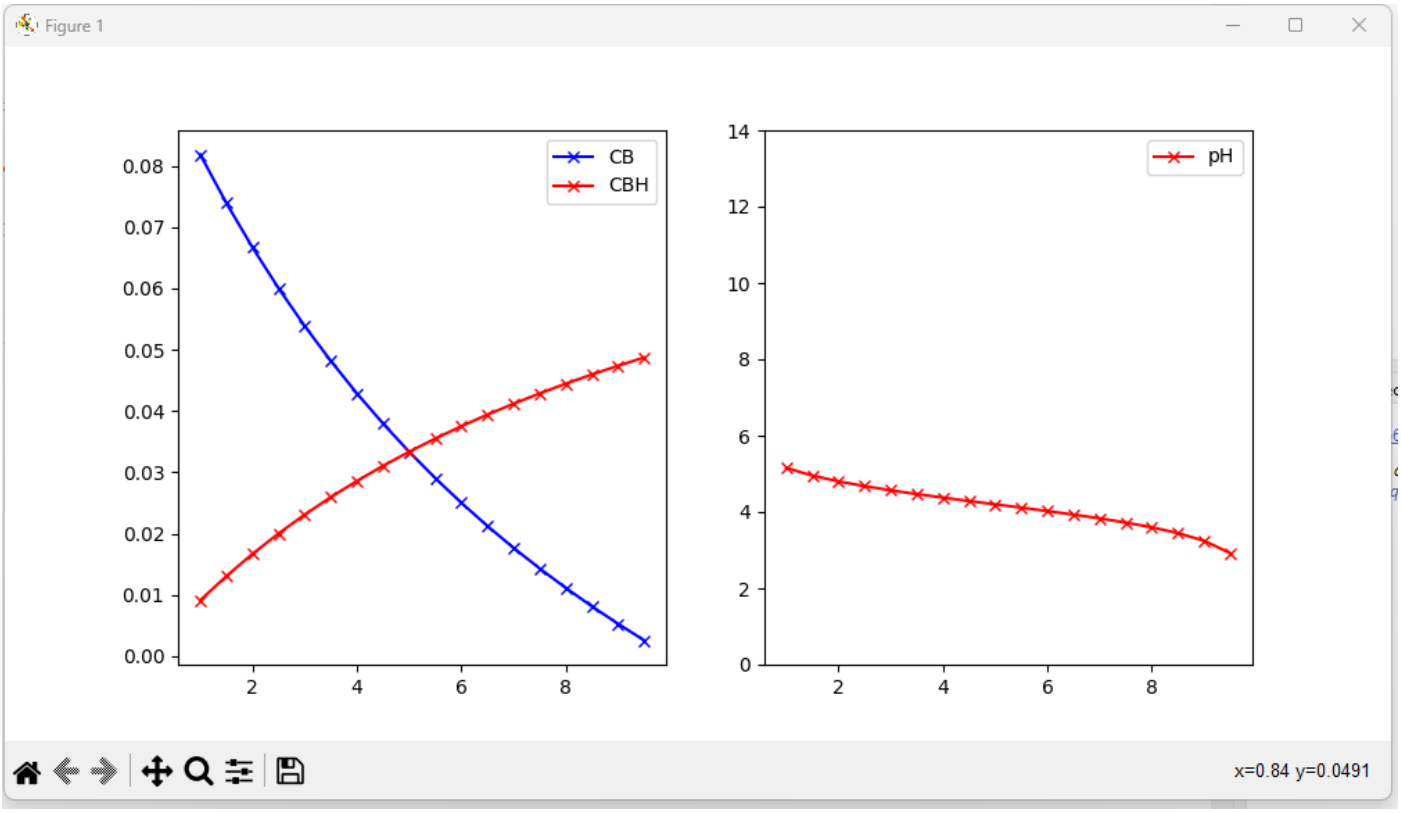
p66ex20.py

```python
import numpy as np
import matplotlib.pyplot as plt
from math import log10


CBI = 0.1  # mol.L⁻¹ Concentration initiale en ion benzoate C6H5COO⁻
CA = 0.100 # mol.L⁻¹ Concentration en ion oxonium H3O+ solution titrar
V0 = 10 # en mL volume de la solution titrée de benzoate de sodium
VE = 10 # en mL volume de solution titrante à l'équivalence

# Initialisation de 5 tableaux de concentrations en ion benzoate (cb),
cb, cbh, ch, ph, v = [], [], [], [], []

### Remplissage des 5 listes suivant le volume va ajouté
for va in np.arange(1,26,0.5) :
    if va < VE :
        v.append(va)
        cb.append((CBI*V0 - CA*va) / (V0 + va))
        cbh.append((CA*va / (V0 + va)))
        ch.append(0)
        ph.append(4.2 + log10((0.1*V0 - CA*va) / (CA * va)))
    elif va > VE:
        pass

# Affichage des graphiques
# Ajuste la taille
# Positionner la fenêtre
plt.figure(figsize=(10, 5))
mngr = plt.get_current_fig_manager()
mngr.window.geometry("+300+100")
# Dessiner
plt.subplot(121)
plt.plot(v, cb, 'bx-', label="CB")
plt.plot(v, cbh, 'rx-', label="CBH")
plt.legend()
plt.subplot(122)
plt.ylim(0,14)
plt.plot(v, ph, 'rx-', label="pH")
plt.legend()
plt.show()
```

A Simulation pH-métrique à compléter

```python
from math import log10
import numpy as np
import matplotlib.pyplot as plt
'''concentration initiale en ions
benzoate Cbi et  [H3O+] de la solution
titrante, volume de la prise d'essai
et volume à l'équivalence'''
CBi, CA, V0, Ve = 0.1, 0.1, 10, 10
'''CB, CBH, CH : liste des concentrations
respectives des espèces B-, BH, H3O+ et
du pH et du volume'''
CB, CBH, CH, pH, V = [], [], [], [], []
for VA in np.arange(1,26,0.5) :
    if VA < Ve :
        V.append(VA)
        CB.append((CBi*V0-CA*VA)/(V0+VA))
        CBH.append((CA*VA/(V0+VA)))
        CH.append(0)
        pH.append(4.2 + log10((0.1*V0-CA*VA)\
        /(CA*VA)))
    elif VA> Ve :

plt.subplot(121)
plt.plot(V, CB, 'bx-', label="CB")
plt.plot(V, CBH, 'rx-', label="CBH")
plt.legend()
plt.subplot(122)
plt.ylim(0,14)
'''définition des valeurs max et min des
ordonnées'''
plt.plot(V, pH, 'rx-', label="pH")
plt.legend()
plt.show()
```

ex20p66.py



5.3. Chap 4 Modélisation macroscopique de l'évolution d'un système

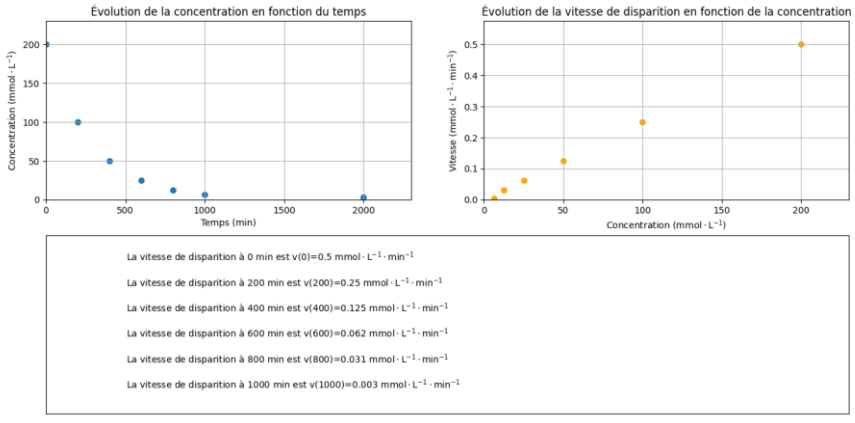
p89ex20.py ×

```python
import matplotlib.pyplot as plt
# Compléter les listes suivantes à partir de l'exercice 13
t = [0, 200, 400, 600, 800, 1000, 2000] # date en min
c = [200, 100, 50, 25, 12.5, 6.3, 3.1] # concentration en mmol.L⁻ ¹
# calcule des vitesses de disparition
vd = []
# pour indice de 1er date à avant dernière calculer vd
for i in range (len(t)-1) :
    vd = vd + [-1*((c[i+1]-c[i]) / (t[i+1]-t[i]))] # On veut une vitesse positive
# recherche des valeurs max pour dimensionner les graphiques
c_max = max(c)
t_max = max(t)
vd_max = max(vd)
# On veut disposer d'un deuxième jeu d'abcisse pour le graphique de vitesse vd = f(concentration)
c_2 = list(c) # On fait une copie (superficielle)
del c_2[-1] # -1 est l'indice du dernier élément
c2_max = max(c_2)
plt.figure(figsize=(16,8))
#sous-figure 1 - Graphique donnant l'évolution de la concentration en fonction du temps
plt.subplot(221) # 2 lignes 2 colonnes 1er cadre
plt.grid()
plt.title("Évolution de la concentration en fonction du temps")
plt.xlim(0, 1.15*t_max) # Pour avoir un graphe plus grand de 15%
plt.ylim(0, 1.15*c_max)
plt.scatter(t, c)        # permet d'afficher un nuage de points, la fonction plt.plot les affiche reliés par défaut
plt.xlabel("Temps (min)")
plt.ylabel(r"Concentration ($\mathrm{mmol \cdot L^{-1}}$)") # Latex
#sous-figure 2- Graphique donnant l'évolution de la vitesse de disparition en fonction du temps
plt.subplot(222) # 2 lignes 2 colonnes 2ieme cadre
plt.grid()
plt.title("Évolution de la vitesse de disparition en fonction de la concentration")
plt.xlim(0,1.15*c2_max)
plt.ylim(0,1.15*vd_max)
plt.scatter(c_2, vd, color='orange')
plt.ylabel(r"Vitesse ($\mathrm{mmol\cdot L^{-1} \cdot min^{-1}}$)")
plt.xlabel(r"Concentration ($\mathrm{mmol\cdot L^{-1}}$)")
#afficher dans la fenêtre (On fait un graphhe vide)
plt.subplot(212, frameon=True)  # 2 ligne 1 colonne cadre ligne 2
plt.xticks([]) # [n] graduation à indiquer ici aucune
plt.yticks([])
for i in range (len(t)-1) :
    plt.text(0.1,
             ((len(t)-1)-i)/len(t),
             "La vitesse de disparition à " +
             str(t[i]) +" min est v("+str(t[i]) +
             ")=" +
             str(round(vd[i],3)) +
             r"$\mathrm{mmol\cdot L^{-1} \cdot min^{-1}}$",
             fontsize = 10 )
plt.show()
```

A Extract from the Python program

```python
import matplotlib.pyplot as plt
import numpy as np
'''compléter les listes suivantes et les décommenter'''
# t = [...]
# C = [...]
```

p89ex20.py

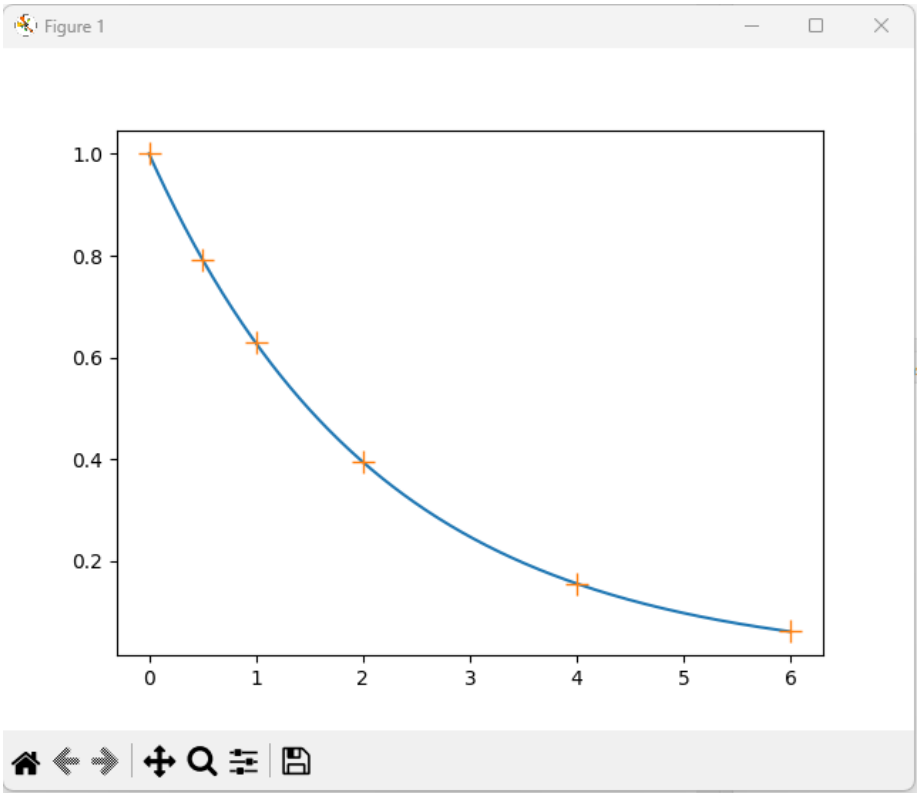


p90ex23.py

```python
import matplotlib.pyplot as plt
# ecart entre deux dates de mesure
DELTA_T =  0.01
# Nombre de point de mesure
N = 600
# intialiser un tableau de N dates
t = [i*DELTA_T for i in range(N)]
# Intialiser une tableau de concentration nulles
c = [0] * N
# Définir la première concentration
c[0] = 1.000 # en mol.L⁻¹
K = 0.464 # Coef en h⁻¹
# Pour indice date de e la première à l'avant dernière
for i in range(N-1):
    c[i+1] = c[i] - (t[i+1] - t[i]) * K * c[i]
# On prepare le tracé
plt.plot(t, c)
# On prépare les mesure réelles
t_mes = [0, 0.5, 1, 2, 4, 6]
c_mes = [1.000, 0.793, 0.630, 0.396, 0.155, 0.063]
# On prépare la nouvelle courbe
plt.plot(t_mes, c_mes, '+', markersize=12) # pour agrandir le +
# On trace
plt.show()
```

```python
import matplotlib.pyplot as plt
Delta_t=0.01
# Delta_t est considéré comme petit
N=600
t=[i*Delta_t for i in range(N)]
C=[0]*N
# Données de l'énoncé :
C[0]=1.000
k=0.464
for i in range(N-1):
    C[i+1]=C[i]-(t[i+1]-t[i])*k*C[i]
plt.plot(t,C)
t_mes=[0, 0.5, 1, 2, 4, 6]
C_mes=[1.000, 0.793, 0.630, 0.396, 0.155,\
0.063]
plt.plot(t_mes,C_mes,'+',markersize=12)
```

p90ex23.py



5.4. Chap 5 Modélisation microscopique de l'évolution d'un système

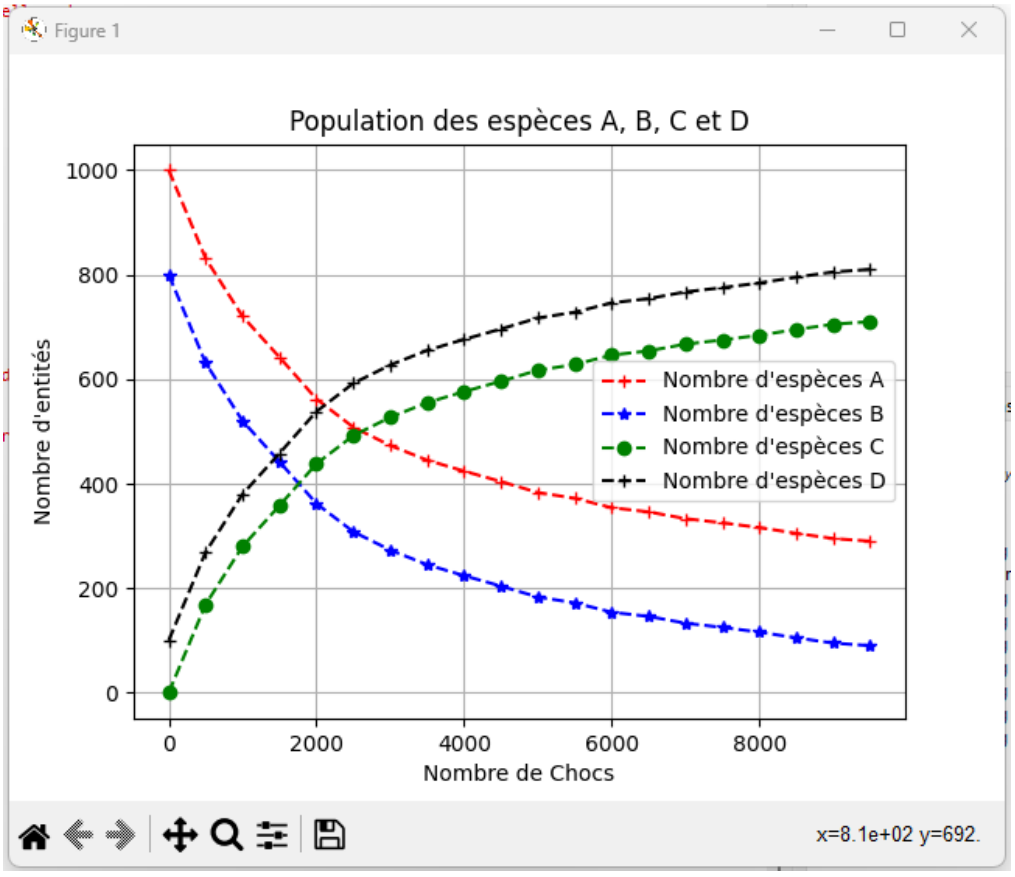
p94act1.py

```python
import matplotlib.pyplot as plt
from random import randint, random

def creation_echantillon(na, nb, nc, nd) :
    '''
    Renvoie un echantillon de na espèce A, n b espèce B, nc espèce C, nd espèce D
    na, nb, nc, nd : int par exemple 1, 2, 3, 4
    echantillon [str]  par exemple ['A', 'B', 'B', 'C', 'C', 'C', 'D', 'D', 'D', 'D']
    '''
    echantillon = []
    for _ind in range(na) :
        echantillon.append("A")
    for _ind in range(nb) :
        echantillon.append("B")
    for _ind in range(nc) :
        echantillon.append("C")
    for _ind in range(nd) :
        echantillon.append("D")
    return echantillon

def comptage_especes(ech, nb_a, nb_b, nb_c, nb_d): # fonction qui compte le nombre d'espèces
    '''
    Dénombre dans l'echantillon  ech le nombre de chaque espèce A, B, C, D
    Ajoute en place ces nombres dans les listes nb_a, nb_b, nb_c, nb_d
    '''
    n_entite = len(ech)
    nbea, nbeb, nbec, nbed = 0,0,0,0
    for j in range(n_entite):
        if ech[j] == "A" :
            nbea = nbea + 1
        elif ech[j] == "B" :
            nbeb = nbeb + 1
        elif ech[j] == "C" :
            nbec = nbec + 1
        elif ech[j] == "D" :
            nbed = nbed + 1
    nb_a.append(nbea)
    nb_b.append(nbeb)
    nb_c.append(nbec)
    nb_d.append(nbed)

def collision(pac, collision_ab) : # fonction qui génère des collisions et remplace A, B éventuellement
    n_entite = len(pac) # nombre d'entité dans le sac
    # définir deux indices différents
    n1 = randint(0, n_entite-1)
    n2 = randint(0, n_entite-1)
    while n2 == n1 :
        n2 = randint(0, n_entite-1)
    # si c'est un choc AB
    if (pac[n1] == "A" and pac[n2] == "B") or (pac[n1] == "B" and pac[n2] == "A") :
        x = random() # au hasard entre
        if x > collision_ab :
            pac[n1], pac[n2] = "C", "D"
    return pac
```

```python
# programme principal
NA = 1000 # nombre d'entités A pouvant être modifié à l'état initial
NB = 800 # nombre d'entités B pouvant être modifié à l'état initial
NC = 0 # nombre d'entités C pouvant être modifié à l'état initial
ND = 100 # nombre d'entités D pouvant être modifié à l'état initial
collisionAB = 0.05 ''' facteur de collision entre A et B compris entre 0 et 1 pouvant être modifié, plus il est grand, moins les chocs sont efficaces
temps = 10000 # variable symbolisant le temps
NbA, NbB, NbC, NbD = [], [], [], [] # création de liste du nombre d'entités A, B, C et D
```

```
55
56  NA = 1000  # nombre d'espèces A
57  NB = 800    # nombre d'espèces B
58  NC = 0  # nombre d'espèces C
59  ND = 100  # nombre d'espèces D
60  COLLISION_AB = 0.05  # facteur de collision entre A et B compris entre 0 et 1, plus il est grand moins les chocs sont efficaces
61  # pas de collision entre C et D
62  temps = 10000  # nombre d'evenements
63  nba, nbb, nbc, nbd = [], [], [], [] # création de liste du nombre d'espèces A, B, C et D au cours du temps
64  sac = creation_echantillon(NA, NB, NC, ND)  # sac une liste
65  t = [0] # liste de temps qui a initialement la valeur 0
66  comptage_especes(sac, nba, nbb, nbc, nbd) # comptage des espèces à t = 0
67
68  for i in range(1, temps) : # a chaque unité de temps
69      collision(sac, COLLISION_AB) # on envisage une collision à chaque unité de temps
70      comptage_especes(sac, nba, nbb, nbc, nbd) # à chaque collision, on compte les espèces
71      t.append(i) # on stocke dans une liste les différents temps
72
73  # réduction du nombre d'éléments pour alléger le tracé
74  K = 500
75  t_light =[t[i] for i in range(len(t)) if i%K == 0]
76  nba_light =[nba[i] for i in range(len(t)) if i%K == 0]
77  nbb_light =[nbb[i] for i in range(len(t)) if i%K == 0]
78  nbc_light =[nbc[i] for i in range(len(t)) if i%K == 0]
79  nbd_light =[nbd[i] for i in range(len(t)) if i%K == 0]
80
81  # affichage des courbes
82  plt.grid()
83  plt.title("Population des espèces A, B, C et D")
84  plt.xlabel("Nombre de Chocs")
85  plt.ylabel("Nombre d'entités")
86  plt.plot(t_light, nba_light, "r--+", label="Nombre d'espèces A") # on prépare la trace Nbre de A en fonction de t
87  plt.plot(t_light, nbb_light, "b--*", label="Nombre d'espèces B") # idem pour B
88  plt.plot(t_light, nbc_light, "g--o", label="Nombre d'espèces C") # idem pour C
89  plt.plot(t_light, nbd_light, "k--+", label="Nombre d'espèces D") # idem pour D
90  plt.legend()
91  plt.show()
```

p94act1.py



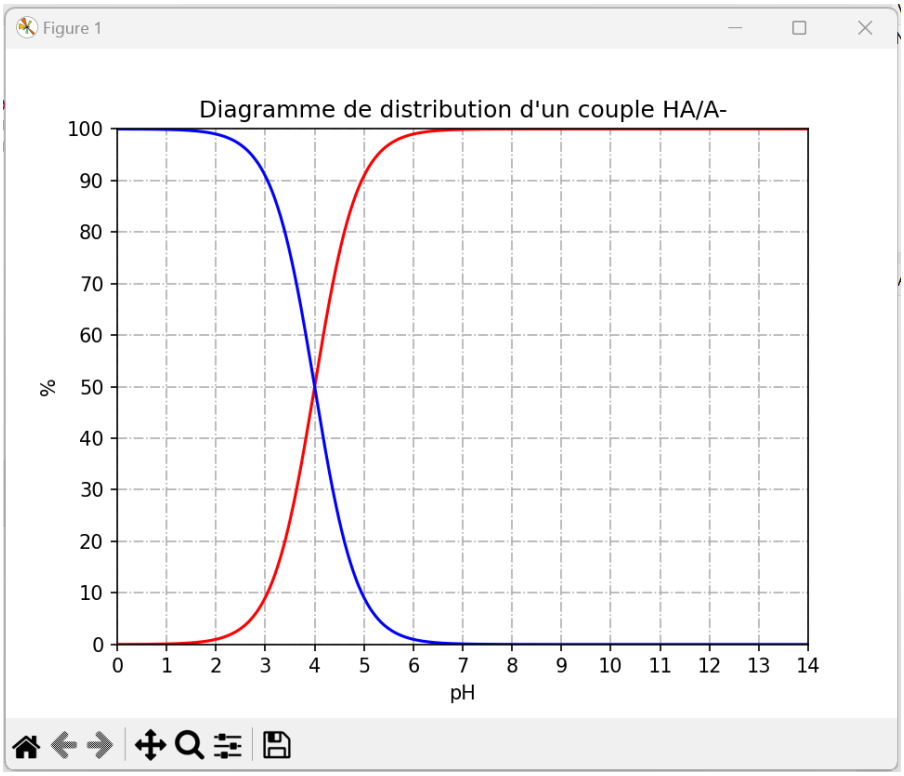
5.5. Chap 8 Force des acides et des bases

p167ex21.py

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  p_ka = float(input('pKA du couple HA / A-  ? '))
4  ph = np.linspace(0, 14, 1000) # Axe linéaire de 1000 valeurs entre 0 et 14
5  ph_ad = [100 / (1 + 10**(i-p_ka)) for i in ph] # Jeu de valeur pH décroissant
6  ph_ac = [100 / (1 + 10**(p_ka-i)) for i in ph] # Jeu de valeur pH croissant
7  plt.title('Diagramme de distribution d\'un couple HA/A-')
8  plt.xlabel('pH')
9  plt.ylabel('%')
10 plt.axis(xmin=0, xmax=14, ymin=0, ymax=100)
11 plt.xticks(range(15))
12 plt.yticks(range(0, 110, 10))
13 plt.grid(linestyle="-.")
14 plt.plot(ph, ph_ac, color='r', label='% en HA' )
15 plt.plot(ph, ph_ad, color='b',label='% en A-' )
16 plt.show()
```

p167ex21.py

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  pKA=float(input('pKA du couple HA / A-  ?'))
4  pH = np.linspace(0.14,1000)
5  pHA = [100/(1+10**(i-pKA)) for i in pH]
6  pA = [100/(1+10**(pKA-i)) for i in pH]
7  plt.title("Diagramme de distribution d'un\
   couple HA/A-")
8  plt.xlabel('pH')
9  plt.ylabel('%')
10 plt.axis(xmin=0,xmax=14,ymin=0, ymax=100)
11 plt.xticks(range(15))
12 plt.yticks(range(0,110,10))
13 plt.grid(linestyle="-.")
14 plt.plot(pH, pHA, color='r', label='% en\
   HA')
15 plt.plot(pH, pA, color='b',label='% en A-')
16 plt.legend()
17 plt.show()
```



p167ex24.py

```
1  ph = float(input('pH de la solution ? pH = ')) # pH de la solution
2  c = float(input('Concentration en soluté apporté en mol.L⁻¹ ? C = '))
3  v = float(input('Volume de la solution en L ? V = '))
4  xf = 10**(-ph*v)
5  xmax = c * v
6  tau = round(xf/xmax, 2) # arrondi pour l'affichage
7  if tau > 1 :
8      print("tau d'avancement = ", tau, "impossible")
9  elif tau == 1:
10     print("tau d'avancement = ", tau, "Acide fort")
11 else:
12     print("tau d'avancement = ", tau, "Acide faible")
13
```

```
1  pH=float(input('pH de la solution ? pH ='))
2  C=float(input('Concentration en soluté apporté en mol/L ? C ='))
3  V=float(input('Volume de la solution en L V='))
4  xf=10**-pH*V
5  xmax=C*V
6  tau=round(xf/xmax,2)
7  if tau>1 : print('taux d'avancement =', tau,'impossible')
8  else :
9    if tau==1 : print('taux d'avancement =',tau,'Acide fort')
10   else :
11     print('taux d'avancement=',tau,'Acide faible')
```

p167ex24.py

Rendu dans la console :

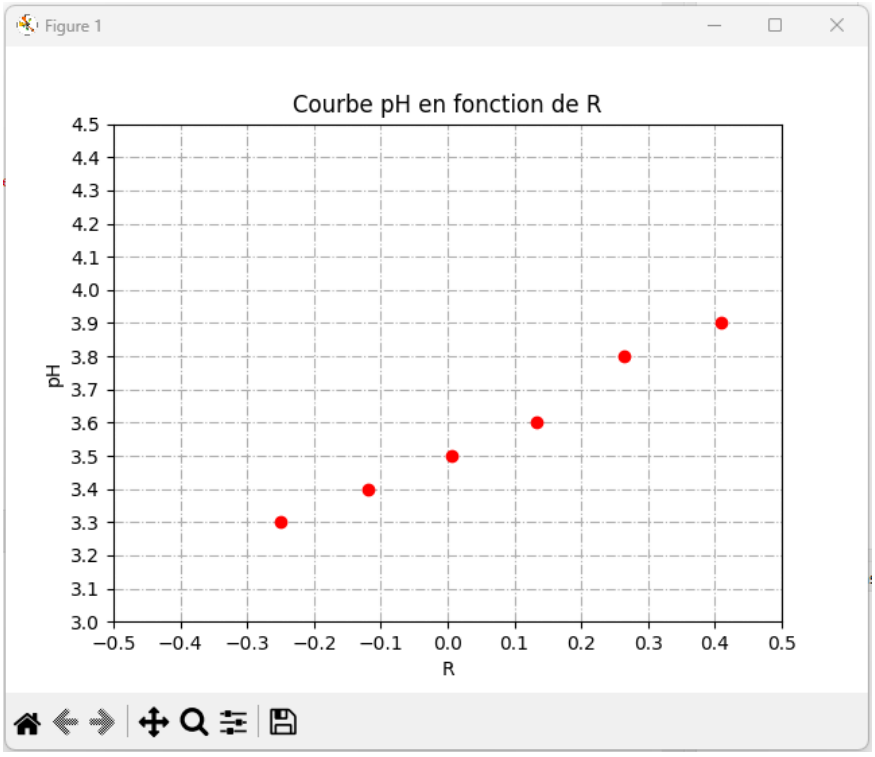
```
>>> %Run p167ex24.py
 pH de la solution ? pH = 1.3
 Concentration en soluté apporté en mol.L⁻¹ ? C = 10.5
 Volume de la solution en L ? V = 0.100
 tau d'avancement =  0.71 Acide faible

>>>
```

p171ece.py

```python
import matplotlib.pyplot as plt
from numpy import log10, arange

# Concentration en hydroxyde de sodium dans la burette
CB = 0.10 # en mol.L⁻¹
# Masse d'acide acétylsalycilique
MA = 500 # en mg
# Masse molaire d'acétylsalycilique
M = 180 #en g.mol⁻¹
# Mesures
v_b = [0.010, 0.012, 0.014, 0.016, 0.018, 0.020] # en L volume ajoutés par burette
ph = [3.3, 3.4, 3.5, 3.6, 3.8, 3.9] # pH de la solution
# calculs des coefficents r
r = [] # coefficient R
for v in v_b :
    r.append(log10(CB*v  / (MA*1E-3/M - CB*v)))
# tracé
plt.title('Courbe pH en fonction de R')
plt.xlabel('R')
plt.ylabel('pH')
plt.axis(xmin=-0.5, xmax=0.5, ymin=3, ymax=4.5)
plt.xticks(arange(-0.5, 0.6, 0.1)) # 1 graduation
plt.yticks(arange(3, 4.6, 0.1))
plt.grid(linestyle="-.")
plt.plot(r, ph, 'ro')
plt.show()
```

```python
import matplotlib.pyplot as plt
from numpy import log10, arange
Vb = [                                    ]
pH = [                                    ]
R=[]
for i in Vb :
    R.append(log10(0.1*i/(0.5/180-0.1*i)))
plt.title('Courbe pH en fonction de R')
plt.xlabel('R')
plt.ylabel('pH')
plt.axis(xmin=-0.5, xmax=0.5, ymin=3, ymax=4.5)
plt.xticks(arange(-0.5,0.6,0.1))
plt.yticks(arange(3,4.6,0.1))
plt.grid(linestyle='-.')
plt.plot(R, pH, 'ro')
plt.show()
```

p171ece.py



5.6. Chap 11 Mouvement et deuxième loi de Newton

p232ex24.py

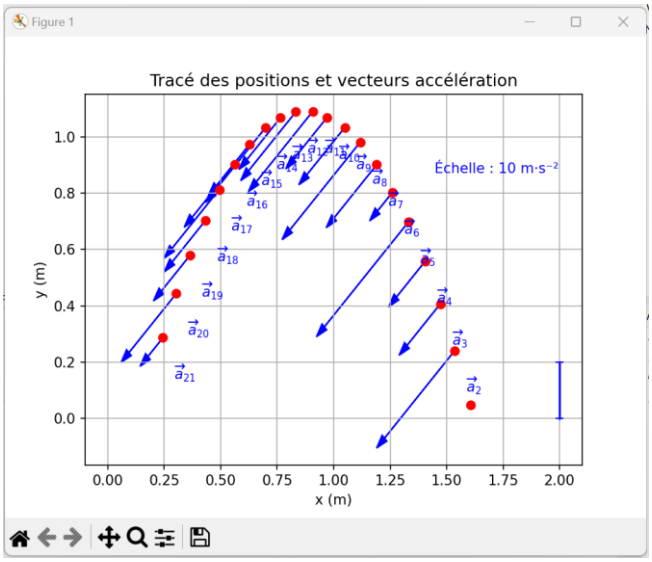
```python
import csv
import matplotlib.pyplot as plt

def import_csv (cible, sep, n):
    '''
    Renvoyer les données d'une colonne d'un fichier csv de séparateur choisi.
    cible : str chemin absolu ou relatif du fichier"
    sep : str le séparateur utilsé (; ou ,)
    n :  int la colonne souhaitée ( à partir de 0)
    rep : liste [ [float]...] les valeurs numériques de la colonne sélectionnée
    '''
    rep = []
    with open(cible, 'r', encoding='utf-8', newline='') as fichier:
        fichier.readline() # elimine la première ligne des noms des champs si nécessaire
        reader = csv.reader(fichier, delimiter=sep)
        for row in reader:
            notation_point = row[n].replace(",",".") # changer le format des nombres "3,5" "3.5"
            rep.append(float(notation_point)) # Transformer en nombre les str
    return rep

K = 0.02 # Facteur echelle pour les accélérations
# Nom du fichier à traiter son chemin csv\balle_de_tennis §attention \ pour les chemins et non /
# m_fichier = input("Quel est le nom du fichier de pointage (sans l'extension .csv)? ") + ".csv"
m_fichier = r"csv\balle_de_tennis.csv"
# Lire les données.
t = import_csv(m_fichier, ";", 0) # la date en s
x = import_csv(m_fichier, ";", 1) # La position abcisse en m
y = import_csv(m_fichier, ";", 2) # La position ordonnée en m

```

```python
'''Calcul des coordonnées vx et vy des vecteurs vitesse'''
vx=[ ]
for i in range(len(x)-1) :
    vx=vx+[(x[i+1]-x[i])/(t[i+1]-t[i])]

vy=[ ]
for i in range(len(y)-1) :
    vy=vy+[(y[i+1]-y[i])/(t[i+1]-t[i])]

'''Calcul des coordonnées ax et ay des vecteurs accélération'''
ax=[ ]
for i in range(len(vx)-1) :
    ax=

ay=[ ]
for i in range(len(vy)-1) :
    ay=
```

p232ex24.py

```python
# Calcul des coordonnées vx et vy des vecteurs vitesse
vx = []
for i in range(len(x)-1) :
    vx = vx + [(x[i+1]-x[i]) / (t[i+1]-t[i])]
vy = []
for i in range(len(y)-1) :
    vy = vy + [(y[i+1]-y[i]) / (t[i+1]-t[i])]
# Calcul des coordonnées ax et ay des vecteurs accélération
ax = []
for i in range(len(vx)-1) :
    ax =  ax + [(vy[i+1]-vy[i]) / (t[i+1]-t[i])]
ay = []
for i in range(len(vy)-1) :
    ay =  ay + [(vy[i+1]-vy[i]) / (t[i+1]-t[i])]

# Légende
plt.title("Tracé des positions et vecteurs accélération")
plt.xlabel('x (m)')
plt.ylabel('y (m)')
plt.grid()
plt.text(0.9*max(x), 0.8*max(y), f"Échelle : {10} m·s⁻²",  color="blue")
plt.plot([2, 2],[0, 10*K], '_-', color='blue') # trace un trait d'échelle
# Tracé
for i in range(len(t)-1):
    plt.plot(0, 0, x[i], y[i],'ro') # position
    # vecteurs accélération
    if i < len(t)-2:
        plt.arrow(x[i+1] , y[i+1],
                  ax[i]*K, ay[i]*K, #appliquer le facteur echelle
                  color='b',
                  head_width=0.03,
                  head_length=0.05,
                  length_includes_head=True)
        plt.text(x[i+1]+0.05, y[i+1]-0.15,
                 r'$ \overrightarrow{a}_{%.i}$'%(i+2),
                 color="blue")
plt.show()
#sauvegarde sous forme d'image
plt.savefig("vecteurs-accélération.png")
```

p232ex24



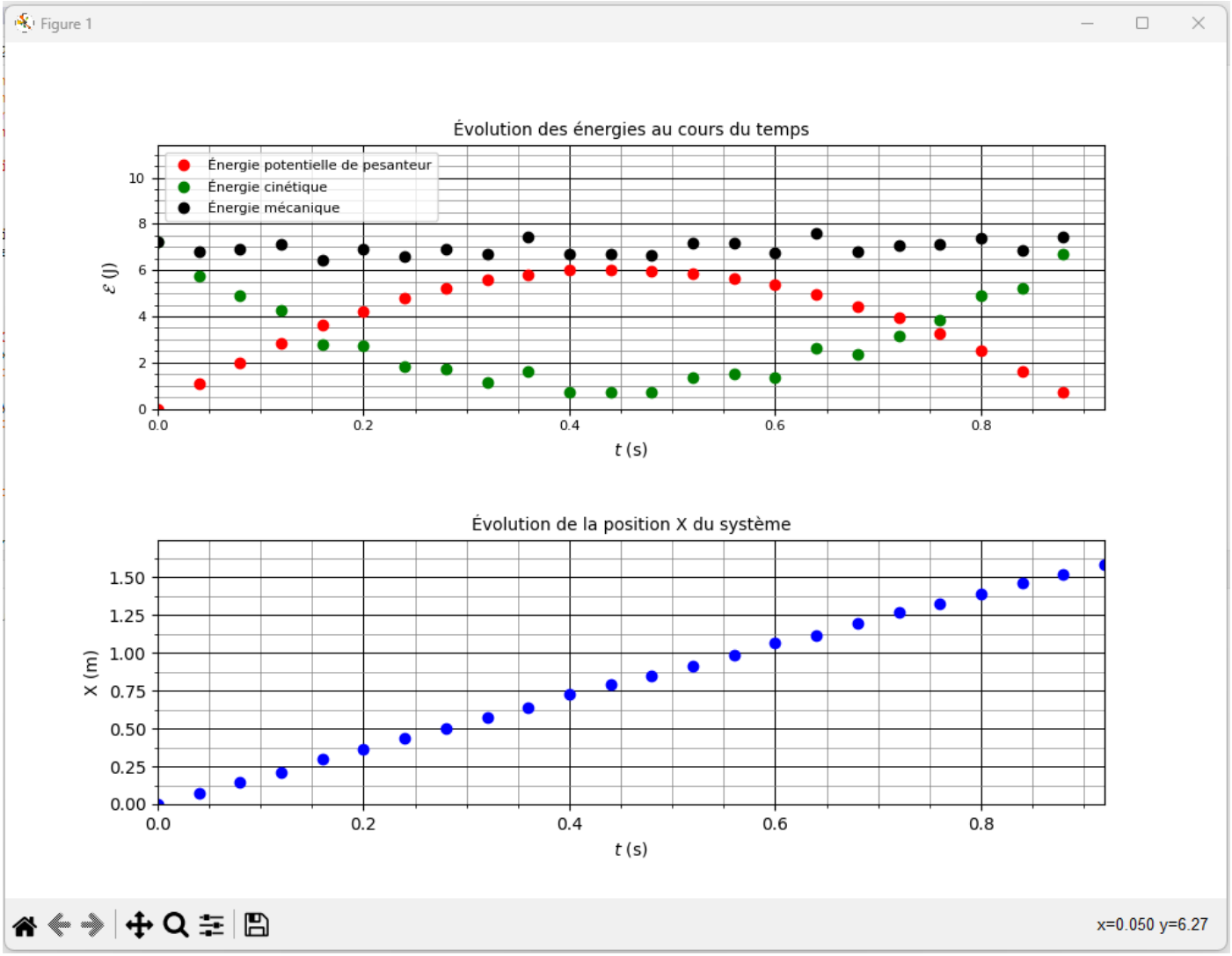
5.7. Chap 12 Mouvement et interaction dans un champ uniforme

p240act2.py

```python
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
#masse du système
M = 0.60 # En kg
#intensité de la pesanteur g
G = 9.81 # en N.kg⁻¹
# Lecture du fichier
# fichier = input("Quel est le nom du fichier de pointage (sans l'extension .csv)?")+".csv"
fichier = "csv\\mouvement_parabolique.csv"
data = pd.read_csv(fichier, sep=';', header=0) # suppose séparateur (;ou,) et nom des champs connus !
# typage (, ->.) et str-> float
t = list(pd.to_numeric(data['Temps'].str.replace(',', '.')))
x = list(pd.to_numeric(data['X'].str.replace(',', '.')))
y = list(pd.to_numeric(data['Y'].str.replace(',', '.')))
#Calcul des coordonnées Vx et Vy des vecteurs vitesse
vx=[]
for i in range(len(x)-1) :
    vx = vx + [(x[i+1]-x[i]) / (t[i+1]-t[i])]
vy=[]
for i in range(len(y)-1) :
    vy = vy + [(y[i+1]-y[i]) / (t[i+1]-t[i])]
# calcule de la vitesse
v = []
for i in range(len(vx)) :
    v = v + [(vx[i]**2 + vy[i]**2)**0.5] # racine carrée ou puissance 1/2
# calcul des énergies
energie_p = [] # Energie potentielle
for i in range(len(vx)) :
    energie_p = energie_p + [M*G*y[i]] # y est l'axe vertical orienté vers le haut
energie_c = []  # Energie cinétique
for i in range(len(vx)) :
    energie_c = energie_c + [0.5*M*v[i]**2]
energie_m = [] # Energie mécanique
for i in range(len(vx))  :
    energie_m = energie_m + [energie_c[i] + energie_p[i]]
# Préparation du graphique
# Ajuster la taille et la position de la fenêtre
plt.figure(figsize=(10, 7))
mngr = plt.get_current_fig_manager()
mngr.window.geometry("+50+50")
# # Ajustement du plot
plt.subplots_adjust(hspace=0.5)
# Sous figure supérieure
plt.subplot(211)
plt.ylim([0, 1.5*max(energie_m)])
plt.xlim([0, max(t)])
#Parametres de la grille
axes = plt.gca()
axes.minorticks_on()
axes.grid(which='major', linestyle='-', linewidth='0.7', color='black')
axes.grid(which='minor', linestyle='-', linewidth='0.5', color='grey')
# Attention le temps dispose d'une valeur de plus !
# Tracé des énergies
plt.plot(t[0:-1], energie_p,'ro', label='Énergie potentielle de pesanteur')
plt.plot(t[0:-1], energie_c,'go', label='Énergie cinétique')
plt.plot(t[0:-1], energie_m,'ko', label='Énergie mécanique')
#Légendes
plt.title("Évolution des énergies au cours du temps", fontsize=10)
plt.ylabel(r'$\mathcal{E}$ (J)', fontsize=10)
plt.xlabel('$t$ (s)', fontsize=10)
plt.tick_params(labelsize=8)
plt.legend(loc=2, fontsize="8")
# Sous figure infèrieure
plt.subplot(212)
plt.ylim([0, 1.1*max(x)])
plt.xlim([0, max(t)])
# #Parametres de la grille
axes2 = plt.gca()
axes2.minorticks_on()
axes2.grid(which='major', linestyle='-', linewidth='0.7', color='black')
axes2.grid(which='minor', linestyle='-', linewidth='0.5', color='grey')
axes2.yaxis.set_minor_locator(AutoMinorLocator(2))
#tracé de la position
plt.plot(t, x, 'bo')
#Légendes
plt.title("Évolution de la position X du système", fontsize=10)
plt.ylabel('X (m)', fontsize=10)
plt.xlabel('$t$ (s)', fontsize=10)
plt.tick_params(labelsize=10)
# tracé
plt.show()
```

p240act2.py

```
v=[]
for i in range(len(vx)) :
    v=v+[(vx[i]**2+vy[i]**2)**0.5]
#masse du système en kg
m=0.60
#intensité de la pesanteur
g=9.81
#Calcul des énergies
Ep=[]
for i in range(len(vx)) :
    Ep=Ep+[m*g*y[i]]
Ec=[]
for i in range(len(vx)) :
    Ec=
Em=[]
for i in range(len(vx)) :
    Em=Em+[Ep[i]+Ec[i]]
plt.show()
```



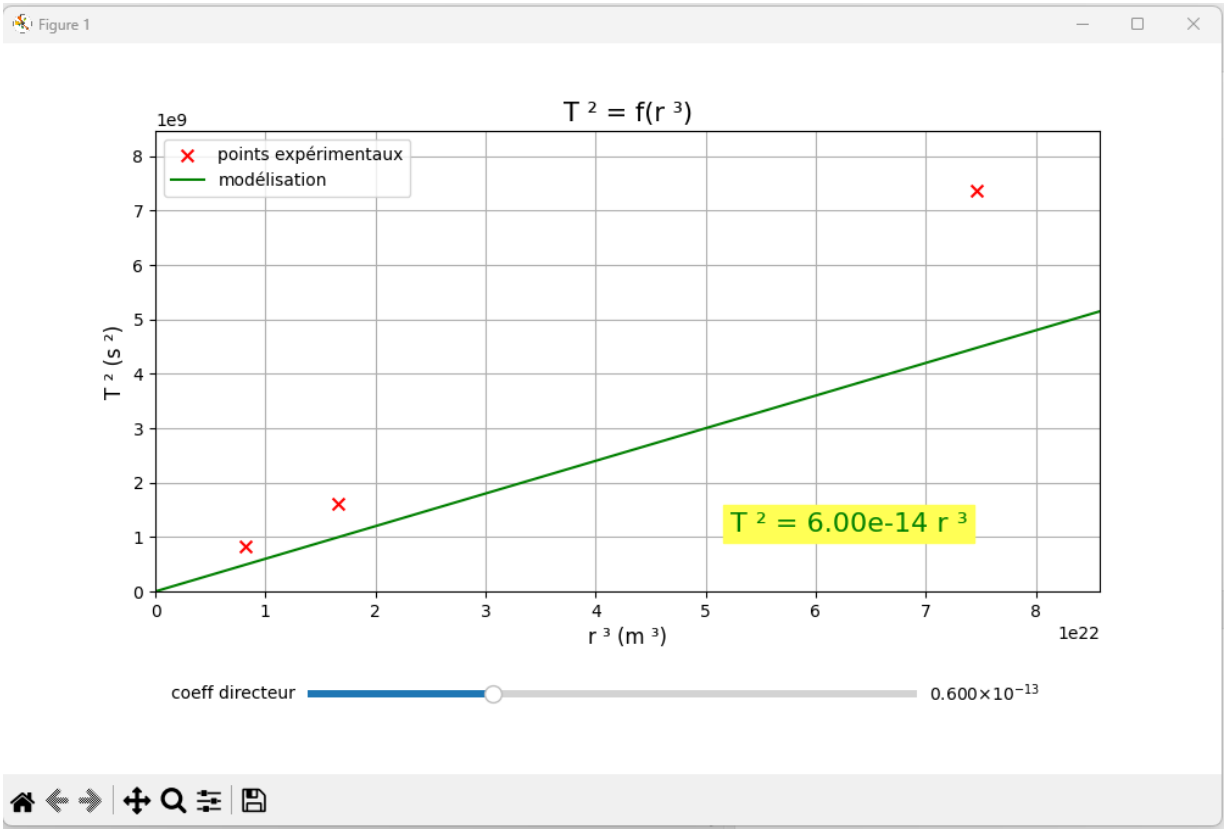
5.8. Chap 13 Mouvement dans un champ de gravitation

p273ex18.py

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from matplotlib.widgets import Slider

def update(val):
    '''
    Actualiser la droite de modélisation et son équation,
    val float la valeur du curseur
    modifie en place
    periode_ma (numpy.ndarray)
    init_a[0] float
    '''
    periode_aj = val * rayon_ma # Nouvelle equation y = ax
    modele[0].set_ydata(periode_aj)
    # le tracé est mis à jour en arrière plan
    # il faut mettre à jour l'objet equation
    equation.set_text(f'T² = {val:.2e} r³')

# Obtention des données type csv
fichier = 'csv\\Satellites_artificiels.csv'
data = pd.read_csv(fichier, sep=';', header=0)
# Mise en forme des données, periode et rayon sont des objets pandas !
periode = data['T(s)']**2 # On conserve T²
rayon = (data['a(km)']*1E3)**3 # On conserve r³
# Calcul de la régression linéaire f sur T² = f(r³)
regression = np.polyfit(rayon, periode, 1) # renvoie les coefficients a, b d'un modèle ax+b
pente_moyenne = regression[0]
# initalisation d'un modèle ajustable y = ax
init_a = 6.00e-14 #pente initiale du modèle
rayon_ma = np.linspace(0, rayon[2]*2, 5) # rayon du modele ajustable
periode_ma = init_a * rayon_ma # periode du modele ajustable

# Tracé des points expérimentaux et de la régression linéaire
# taille et position de la fenêtre
plt.figure(figsize=(10,6))
plt.get_current_fig_manager().window.geometry("+5+5")
# Ajustement du plot
plt.subplots_adjust(bottom=0.25) #on laisse un espace pour le curseur
# définition des axes
plt.axis([0, max(rayon)*1.15, 0, max(periode)*1.15])
plt.grid() # Paramètre de la grille
# légendes des axes
plt.xlabel('r ³ (m ³)', fontsize=12)
plt.ylabel('T ² (s ²)', fontsize=12)
plt.tick_params(labelsize=10)
plt.title('T ² = f(r ³)', fontsize=15)
# Tracé des points de mesure
plt.scatter(rayon, periode, s=50, color='r', marker='x', label='points expérimentaux')
# Affichage du modèle ajustable, modele[0] de classe matplotlib.lines.Line2D
modele = plt.plot(rayon_ma, periode_ma, '-g', label='modélisation')
plt.legend(loc=2, fontsize=10) #Légende du graphique
# affichage de l'équation dans le graphique
equation = plt.text(max(rayon)*0.7,max(periode)*0.15,
                    f'T ² = {init_a:.2e} r ³',
                    fontsize = 16, color="green", backgroundcolor = "#FFFF55")
# Création d'un curseur
rectangle_a = plt.axes([0.25, 0.1, 0.5, 0.02])
curseur = Slider(rectangle_a, 'coeff directeur', 0, pente_moyenne*2, valinit=init_a)
# appel update lorsque le curseur est modifié
curseur.on_changed(update)
plt.show() # tracé en boucle
```

p273ex18.py

```python
#Variables et régression linéaire
#Calcul des variables associées à la fonction
Tcarre = T**2
rcube = (r*1000)**3
#Calcul de la régression linéaire
regression = np.polyfit(rcube, Tcarre,1)
pente_moyenne=regression[0]
init_a=6.00e-14
#Variables pour la régression
rcube_regression=np.linspace(0,rcube[2]*2,5)
T_regression = init_a*rcube_regression
```



5.9. Chap 14 Modélisation de l'écoulement d'un fluide

p297syn.py

```python
import numpy as np

# angle de lancement par rapport à l'horizontale
V0 = 25 # en degre
# intensité de la pesanteur g
G = 9.81 # en m·s⁻²

# angle de lancement converti de degré en radian
alpha = np.radians(V0)
# calcul de la durée maximale
tmax = (2 * V0 * np.sin(alpha) / G) * 1.1 # Depassement de 10%
# calcul des dates
t = np.linspace(0, tmax, 10)
# calcul des coordonnées x et y des positions
x = V0 * np.cos(alpha) * t
y = -G/(2*(V0 * (np.cos(alpha)))**2) *  x**2 + np.tan(alpha) * x
# calcul des coordonnées vx et vy des vecteurs vitesse
vx=[]
for i in range(len(x)-1) :
    vx = vx + [(x[i+1]-x[i]) / (t[i+1]-t[i])]
vy=[]
for i in range(len(y)-1) :
    vy = vy + [(y[i+1]-y[i]) / (t[i+1]-t[i])]
# calcul des coordonnées ax et ay des vecteurs accélération
ax = []
for i in range(len(vx)-1) :
    ax = ax + [(vx[i+1]-vx[i]) / (t[i+1]-t[i])]
ay = []
for i in range(len(vy)-1) :
    ay = ay + [(vy[i+1]-vy[i]) / (t[i+1]-t[i])]

# affichage dans la console
print ("coordonnées successives de la position, (x ; y)")
for i in range (len(x)):
    print(f'({x[i]:+.2E} m ; {y[i]:+.2E} m)') #+ pour le signe .2 pour 2 chiffres et E pour Exposant
print()
print ("coordonnées successives de la vitesse, (vx ; vy)")
for i in range (len(vx)):
    print(f'({vx[i]:+.2E} m·s⁻¹ ; + {vy[i]:+.2E} m·s⁻¹)')
print("")

print ("coordonnées successives de l'accélération, (ax ; ay)")
for i in range (len (ax)):
    print(f'({ax[i]:+.2E} m·s⁻² ; + {ay[i]:+.2E} m·s⁻²)')

```

p297syn.py

B **Modélisation avec un langage de programmation**

Le programme suivant permet de modéliser les positions du ballon lors de cette pénalité.

```python
v0=25
alpha=40*np.pi/180
g=9.81
tmax=2.2*v0*np.sin(alpha)/g
t=np.linspace(0,tmax,10)
x=v0*np.cos(alpha)*t
y=-1/2*g*x**2/(v0**2*(np.cos(alpha))**2)+\
                x*np.tan(alpha)
```

Un extrait du programme permettant de calculer les coordonnées cartésiennes du vecteur vitesse est donné ci-dessous :

```python
vx=[]
for i in range(len(x)-1) :
    vx=vx+[(x[i+1]-x[i])/(t[i+1]-t[i])]

vy=[]
for i in range(len(y)-1) :
    vy=vy+[(y[i+1]-y[i])/(t[i+1]-t[i])]
```

```
>>> %Run p297syn.py

  coordonnées successives de la position, (x ; y)
  (+0.00E+00 m ; +0.00E+00 m)
  (+5.97E+00 m ; +2.44E+00 m)
  (+1.19E+01 m ; +4.20E+00 m)
  (+1.79E+01 m ; +5.28E+00 m)
  (+2.39E+01 m ; +5.69E+00 m)
  (+2.98E+01 m ; +5.41E+00 m)
  (+3.58E+01 m ; +4.45E+00 m)
  (+4.18E+01 m ; +2.81E+00 m)
  (+4.77E+01 m ; +4.94E-01 m)
  (+5.37E+01 m ; -2.50E+00 m)

  coordonnées successives de la vitesse, (vx ; vy)
  (+2.27E+01 m·s⁻¹ ; + +9.27E+00 m·s⁻¹)
  (+2.27E+01 m·s⁻¹ ; + +6.69E+00 m·s⁻¹)
  (+2.27E+01 m·s⁻¹ ; + +4.11E+00 m·s⁻¹)
  (+2.27E+01 m·s⁻¹ ; + +1.53E+00 m·s⁻¹)
  (+2.27E+01 m·s⁻¹ ; + -1.06E+00 m·s⁻¹)
  (+2.27E+01 m·s⁻¹ ; + -3.64E+00 m·s⁻¹)
  (+2.27E+01 m·s⁻¹ ; + -6.22E+00 m·s⁻¹)
  (+2.27E+01 m·s⁻¹ ; + -8.80E+00 m·s⁻¹)
  (+2.27E+01 m·s⁻¹ ; + -1.14E+01 m·s⁻¹)

  coordonnées successives de l'accélération, (ax ; ay)
  (+0.00E+00 m·s⁻² ; + -9.81E+00 m·s⁻²)
  (+0.00E+00 m·s⁻² ; + -9.81E+00 m·s⁻²)
  (+0.00E+00 m·s⁻² ; + -9.81E+00 m·s⁻²)
  (+0.00E+00 m·s⁻² ; + -9.81E+00 m·s⁻²)
  (+0.00E+00 m·s⁻² ; + -9.81E+00 m·s⁻²)
  (+0.00E+00 m·s⁻² ; + -9.81E+00 m·s⁻²)
  (+0.00E+00 m·s⁻² ; + -9.81E+00 m·s⁻²)
  (-2.70E-14 m·s⁻² ; + -9.81E+00 m·s⁻²)
```

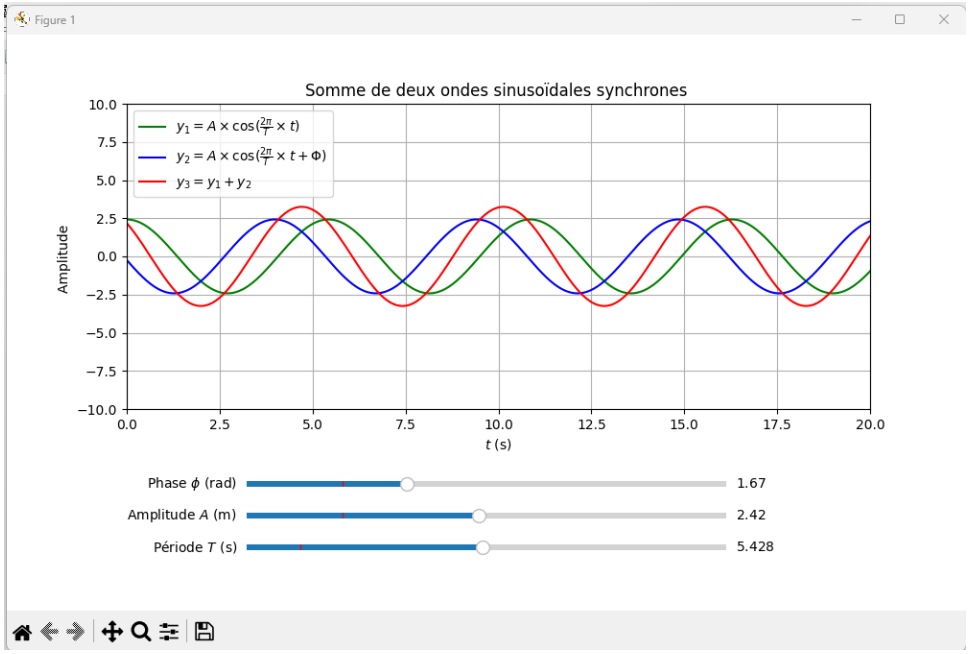
5.10. Chap 18 diffraction et interférence

p381ex22.py

```python
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.widgets import Slider

#Fonction d'actualisation des courbes
def update(_x):
    '''
    Actualiser le graphique
    x float la valeur de l'un des curseurs
    variables globales curseur_A, curseur_T, curseur_PHI
    Modifie en place les tracés p1, p2 et p3
    '''
    # Nouveaux coefficients des objets curseurs
    A = curseur_A.val
    T = curseur_T.val
    PHI = curseur_PHI.val
    # Calcul nouveau jeux de valeurs
    y_1 = A * np.cos(2*np.pi/T * time)
    y_2 = A * np.cos(2*np.pi/T * time + PHI)
    y_3 = y_1 + y_2
    #mis à jour des tracés
    p1[0].set_ydata(y_1)
    p2[0].set_ydata(y_2)
    p3[0].set_ydata(y_3)


#  valeurs par défaut
AMP = 1.00 # Amplitude
PER = 2.00 # Periode
DPHA = 1.00 # Dephasage
# Définitions des courbes
time = np.linspace(0, 20, 2000)
y1 = AMP * np.cos(2*np.pi/PER * time)
y2 = AMP * np.cos(2*np.pi/PER * time + DPHA)
y3 = y1 + y2

#Tracé des courbes
# taille et position de la fenêtre
plt.figure(figsize=(10,6))
plt.get_current_fig_manager().window.geometry("+5+5")
# Ajustement du plot
plt.subplots_adjust(bottom=0.35) #on laisse un espace pour le curseur
# définition des axes
plt.axis([0, 20, -10, 10])
plt.grid() # Paramètre de la grille
# légendes des axes
plt.xlabel('$t$ (s)')
plt.ylabel('Amplitude ')
plt.title('Somme de deux ondes sinusoïdales synchrones ')

# tracé des courbes pn[0] de classe matplotlib.lines.Line2D
p1 = plt.plot(time, y1, '-g',label=r'$y_1 = A \times \cos( \frac{2\pi}{T}\times t)$')
p2 = plt.plot(time, y2, '-b',label=r'$y_2 = A \times \cos( \frac{2\pi}{T}\times t + \Phi)$')
p3 =  plt.plot(time, y3, '-r',label=r'$y_3= y_1 + y_2$')
plt.legend(loc=2, fontsize=10) #Légende du graphique


# Création des curseurs
# Curseur periode
rectangle_a = plt.axes([0.25, 0.1, 0.5, 0.02])
curseur_T = Slider(rectangle_a, 'Période $T$ (s)', 1, 10, valinit=PER)
# Curseur amplitude
rectangle_b = plt.axes([0.25, 0.155, 0.5, 0.02])
curseur_A = Slider(rectangle_b, 'Amplitude $A$ (m)', 0, 5, valinit=AMP)
# Curseur phase
rectangle_c = plt.axes([0.25, 0.210, 0.5, 0.02])
curseur_PHI = Slider(rectangle_c, r'Phase $\phi$ (rad)', 0, 5, valinit=DPHA)

# appel update lorsque l'un des curseurs est modifié
curseur_T.on_changed(update)
curseur_A.on_changed(update)
curseur_PHI.on_changed(update)

plt.show()
```

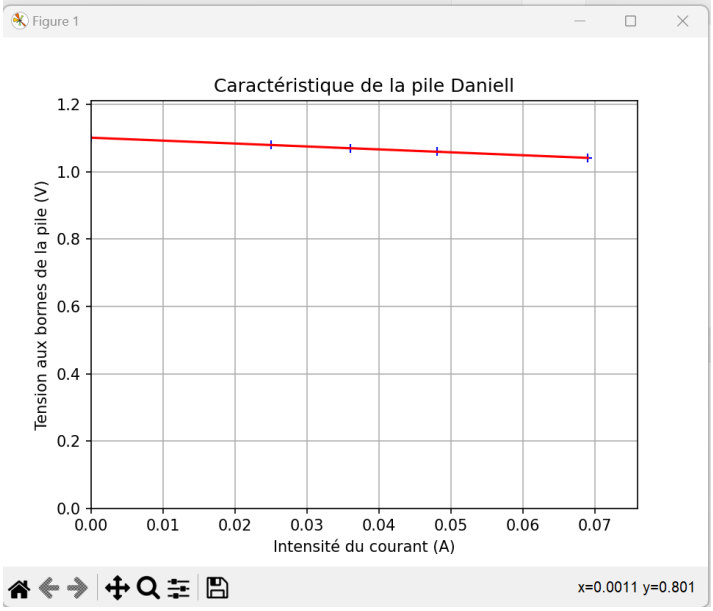
p381ex22.py

```
#Conditions initiales
initial_A = 1.00
initial_T = 2.00
initial_Phi = 1.00
# Définitions des courbes
time=np.linspace(0.,20.,2000)
y1=initial_A*np.cos(2*np.pi/initial_T*time)
y2=initial_A*np.cos(2*np.pi/initial_T*\
time+initial_Phi)
y3=y1+y2
#Tracé des courbes
G = GridSpec(10, 10)
fig, ax = plt.subplots()
axes_1 = plt.subplot(G[:-3, :])
plt.axis([0,20,-10,10])
plt.xlabel('t en s')
plt.ylabel('Amplitude')
plt.title('Somme de deux ondes sinusoïdales \
synchrones')
plt.grid()
p1, = plt.plot(time, y1, '-g' , label=\
r'$y_1=A\times\cos( \frac{2\pi}{T}\times t)$')
p2, = plt.plot(time, y2, '-b', label=\
r'$y_2 =A\times\cos( \frac{2\pi}{T}\times t+\
Phi)$')
p3, = plt.plot(time, y3, '-r',label=\
r'$y_3= y_1 + y_2$')
plt.legend()
```



6. Exercices

6.1. Tracé de la caractéristique d'une pile

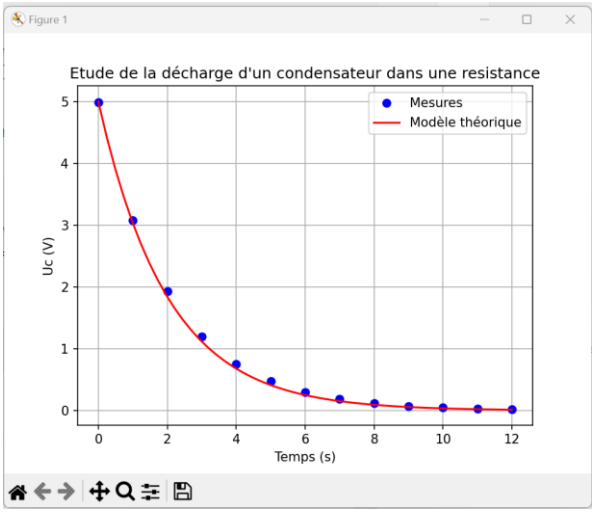


ex6_1.py

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

# Points expérimentaux
intensite = [0, 0.025, 0.036, 0.048, 0.069] # en A
tension = [1.1, 1.08, 1.07, 1.06, 1.04] # en V

# A compléter
```

6.2. Tracé de la décharge d'un condensateur

On veut obtenir un graphique à partir de mesures, puis placer sur ces points un modèle mathématique connu.

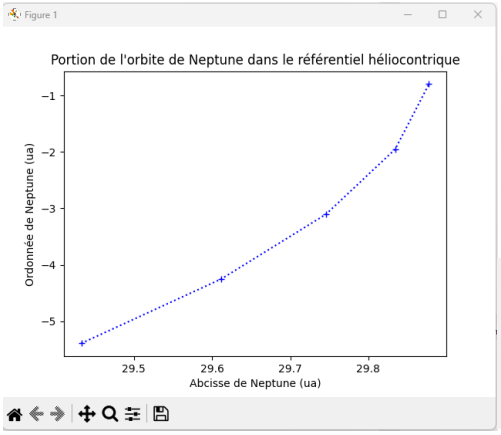


ex6_2.py

```
# Tracé des courbes théoriques et expérimentales
# de la décharge d'un condensateur de 100 µF
# dans un conducteur ohmique de 20kΩ

import matplotlib.pyplot as plt
import numpy as np

C = 100E-6 # en farad
R = 20E3 # en ohm
UINIT = 5 #en volt

# valeurs expérimentales
# date en seconde
t_ex = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
                 10, 11, 12])
# tensions aux bornes du condensateur en volt
uc_ex = np.array([4.99, 3.08, 1.93, 1.2, 0.75,
                  0.47, 0.3, 0.19, 0.12, 0.07, 0.05,
                  0.03, 0.02])
# A compléter
```

6.3. Vérification de la deuxième loi de Kepler

On veut vérifier que des mesures vérifient une loi connue.



Shell

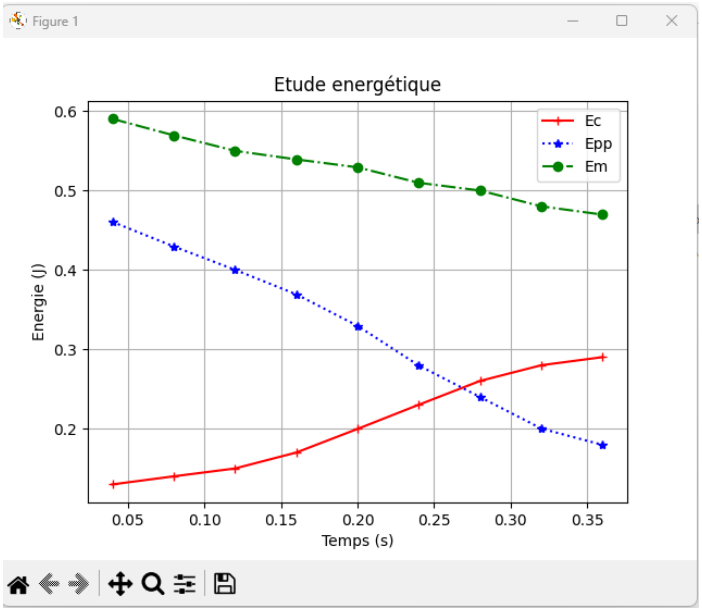
```
>>> %Run ex6_3.py

  l'aire parcourue par le rayon vecteur entre 2020 et 2021 vaut 17.26 ua²
  l'aire parcourue par le rayon vecteur entre 2022 et 2023 vaut 17.26 ua²
  Neptune vérifie la deuxième loi de Kepler

>>>
```

ex6_3.py

```python
# Vérification de la deuxième loi de Kepler
import matplotlib.pyplot as plt
import math as m

# valeurs expérimentales
# date en année
t_ex = [2020, 2021, 2022, 2023, 2024]
# Coordonnées en unité astronomique ua
x = [29.4328410560730, 29.6118356847488,
     29.74576820311831, 29.8342404541355,
     29.8774316380940] # abcisse
y = [-5.3906071859853, -4.2505317595919,
     -3.10349008723970,-1.9519995345194,
     -0.7985426911486] # ordonnées

def calcul_aire(an):
    '''
    renvoyer l'aire parcourue
    '''
    i = t_ex.index(an)
    # calcul des aires 2020-2021
    long1 = m.sqrt(x[i]**2 + y[i]**2)
    long2 = m.sqrt(x[i+1]**2 + y[i+1]**2)
    long3 = m.sqrt((x[i+1]-x[i])**2 + (y[i+1]-y[i])**2)
    demip = (long1 + long2 + long3) / 2 # demi perimetre
    aire = m.sqrt(demip * (demip - long1) * (demip - long2) * (demip - long3))
    return aire
# A compléter
```

6.4. Etude énergétique

On veut représenter sur un même graphique, plusieurs séries de valeurs, calculées à partir de données expérimentales.

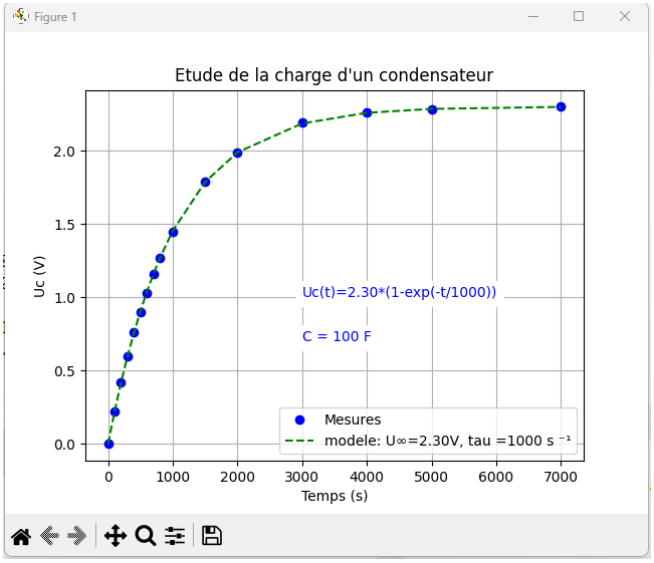


ex6-4.py

```
# Représenter l'évolution au cours du temps des énergie cinétique, potentielle de pesanteur et mécanique
# d'un grélon en chute libre

import matplotlib.pyplot as plt

M = 15E-3  # masse du grelon en kg
G = 9.81 # Intensité de la pesanteur en N.kg⁻¹
# Mesures expérimentales
z = [3.13, 2.92, 2.72, 2.51, 2.24, 1.90, 1.63, 1.36, 1.22] # altitude en m
t = [0.04, 0.08, 0.12, 0.16, 0.20, 0.24, 0.28, 0.32, 0.36] # date e s
v = [4.16, 4.32, 4.47, 4.76, 5.16, 5.54, 5.89, 6.11, 6.22] # vitesse en m.s⁻¹
# A compléter
```

6.5. Obtenir la loi de charge d'un condensateur

A partir de données expérimentale, ajuster un modèle mathématique, en déduire la valeur d'un composant.



ex6_5P.py

```python
# Tracé des courbes expérimentale de la charge d'un condensateur
# avec un conducteur ohmique de 10Ω

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

R = 10 # en ohm

# valeurs expérimentales
# date en seconde
t_ex = np.array([0, 100, 200, 300, 400, 500, 600, 700, 800,
                 1000, 1500, 2000, 3000, 4000, 5000, 7000])
# tensions aux bornes du condensateur en volt
uc_ex = np.array([0, 0.22, 0.42, 0.6, 0.76, 0.9, 1.03, 1.16,
                  1.27, 1.45, 1.79, 1.99, 2.19, 2.26, 2.28, 2.3])

def modele(x, a, b):
    return a * (1 - np.exp(-x/b))

u_inf = max(uc_ex)
# p0 initialisation nécessaire des valeurs des parametres si le modèle n'est pas trouvé
(params, covariance) = curve_fit(modele, t_ex, uc_ex, p0 = [u_inf, R])
# A compléter
```