

Culture Informatique

3 Heures Python/Arduino

Table des matières

1.	Langages de programmation	3
1.1.	Qu'est-ce qu'un langage de programmation ?	3
1.2.	Interprétation et compilation	3
1.2.1.	La compilation	3
1.2.2.	L'interprétation.	3
2.	Structuration d'un programme	4
2.1.	Notion de programme	4
2.2.	Notion de « variable »	4
2.3.	Notion d'objet	4
2.4.	Affectation	4
2.5.	Les fonctions	5
2.6.	Muable ou immuable, valeur ou référence	5
2.7.	La structure conditionnelle	6
2.8.	Les structures itératives	7
2.8.1.	Boucle bornée (à privilégier)	7
2.8.2.	Boucle non bornée	7
3.	Objets et méthodes	8
3.1.	Les nombres	8
3.1.1.	Python	8
3.1.2.	Arduino	8
3.2.	Les booléens et les tests	8
3.3.	Les chaînes de caractères	8
3.4.	Les tuples	8
3.5.	Les listes	8
3.6.	Les dictionnaires	8
4.	Modules complémentaires Python	9
4.1.1.	Présent ou pas	9
4.1.2.	Importation du module	9
4.2.	Le module math	9
4.3.	Le module numpy (calcul numérique)	10
4.4.	Le module scipy (calcul scientifique)	10
4.5.	Le module matplotlib (tracé de courbes)	10

- 4.6. Autres modules (random, time, pandas ,...) 10
- 4.7. Lecture et écriture de fichiers 10
- 5. Exercices commentés 11
 - 5.1. Chap 2 Méthodes physiques d'analyse 11
 - 5.2. Chap 3 méthodes chimiques d'analyse 13
 - 5.3. Chap 4 Modélisation macroscopique de l'évolution d'un système 14
 - 5.4. Chap 5 Modélisation microscopique de l'évolution d'un système 16
 - 5.5. Chap 8 Force des acides et des bases 18
 - 5.6. Chap 11 Mouvement et deuxième loi de Newton 21
 - 5.7. Chap 12 Mouvement et interaction dans un champ uniforme 23
 - 5.8. Chap 13 Mouvement dans un champ de gravitation 25
 - 5.9. Chap 14 Modélisation de l'écoulement d'un fluide 27
 - 5.10. Chap 18 diffraction et interférence 28
- 6. Exercices 30
 - 6.1. Tracé de la caractéristique d'une pile 30
 - 6.2. Tracé de la décharge d'un condensateur 31
 - 6.3. Vérification de la deuxième loi de Kepler 32
 - 6.4. Etude énergétique 33
 - 6.5. Obtenir la loi de charge d'un condensateur 34

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.



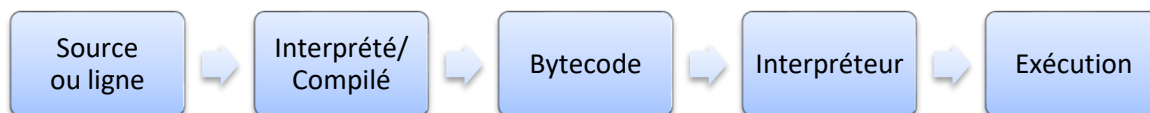
1.2.2. L'interprétation.

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.

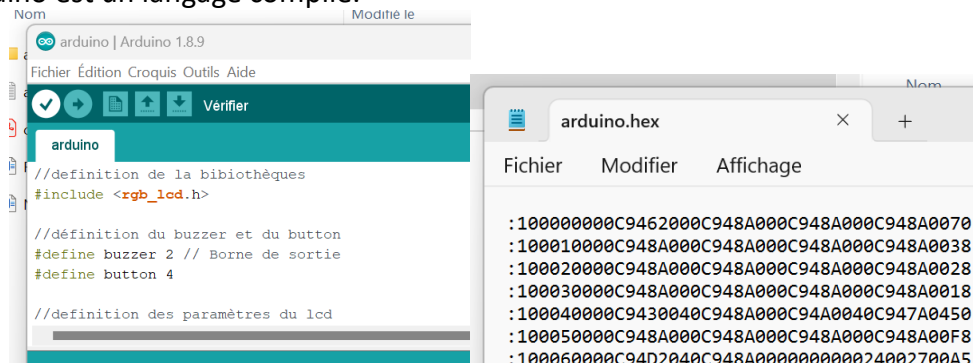


Python est un langage interprété, semi-compilé. On peut ainsi bénéficier

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

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.

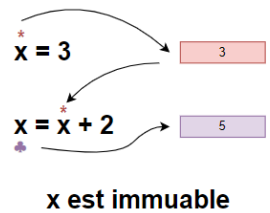
2.6. Muable ou immuable, valeur ou référence

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.

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

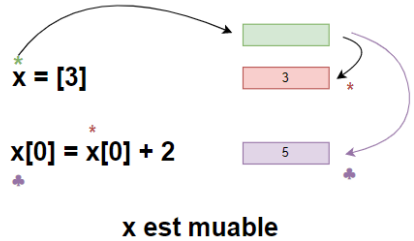
```
>>> nb_un = 32
>>> nb_un = 40
```

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

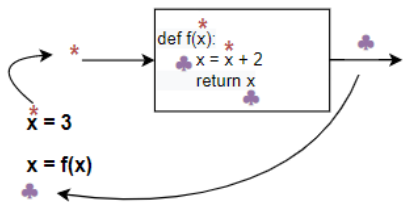


Contrairement aux types intégrés tels que int, float, bool, string, unicode, tuple qui sont des types d'objets immuables, les types list, dict, set sont des types d'objets muables. Les classes personnalisées sont aussi généralement modifiables.

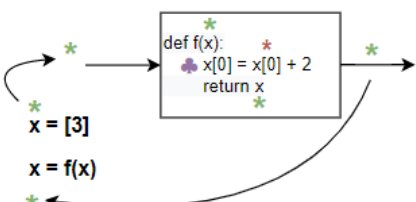
```
>>> x = ["red", "blue", "green"]
>>> id(x)
87074096
>>> x[0] = "pink"
>>> id(x)
87074096
```



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

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
<pre>if a > 5: a = a + 1 else: if a < 3: a = a - 1 else: a = 0</pre>	<pre>if (a > 5) { a = a + 1 ; }else{ if (a < 3) { a = a - 1 ; }else{ a = 0 ; } }</pre>

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

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

pour chaque élément d'un objet itérable

Python	Arduino
<pre>for element in objet_iterable: blocInstructions</pre>	<p>INEXISTANT</p>

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
<p>INEXISTANT</p>	<pre>do { // instruction(s); }while (condition);</pre>

Tant que condition faire action

Python	Arduino
<pre>while cond: instructions</pre>	<pre>while (condition) { // instruction(s); }</pre>

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

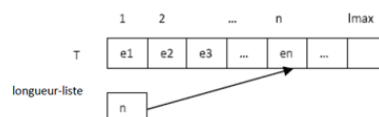
Les tuples sont des n-uplets. Ils sont notés entre parenthèses :

(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 chaînée



- 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 partir 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 précédent)

'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

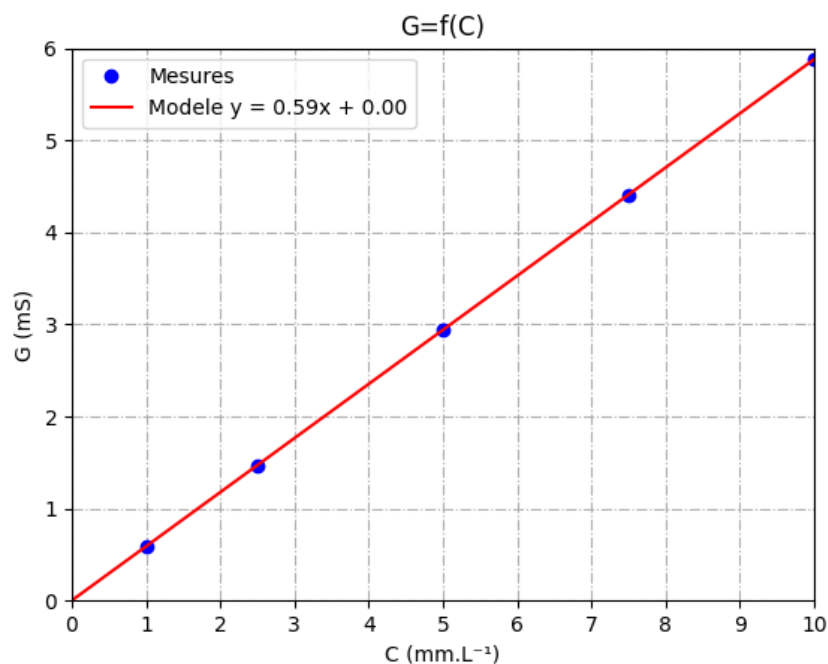
5. Exercices commentés

5.1. Chap 2 Méthodes physiques d'analyse

p45ex21.py

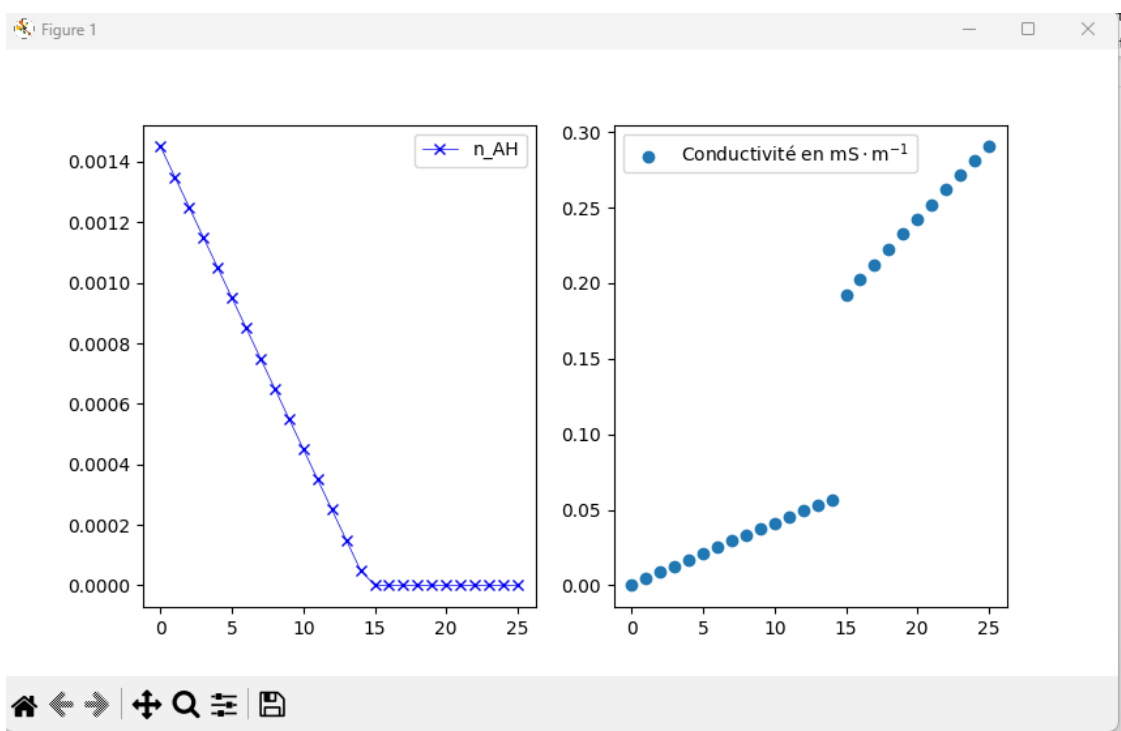
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 # Données expérimentales
4 x = [10.0, 7.5, 5.0, 2.5, 1.0] # C en mm.L-1
5 y = [5.88, 4.41, 2.94, 1.47, 0.59] # G en mS
6 # Affichage
7 plt.title("G=f(C)")
8 plt.xlabel("C (mm.L\u207B\u00B9)") # unicode -1
9 plt.ylabel("G (mS)")
10 plt.plot(x, y, "bo", label='Mesures') # symbole b1
11 plt.legend()
12 plt.axis(xmin=0, xmax=10, ymin=0, ymax=6)
13 plt.grid(linestyle="-.")
14 plt.xticks(range(11)) # nb de lignes
15 # Modélisation polynôme ordre 1
16 coef = np.polyfit(x, y, 1) # renvoie array([0.5878424, 0.00121951])
17 # Calcul du modèle pour 5 valeurs prises entre 0 et xmax
18 xmod = np.linspace(0, max(x), 5)
19 modele = [coef[1] + coef[0] * val for val in xmod] # Calcul du modèle
20 plt.plot(xmod, modele, color = "red", label = f"Modèle y = {coef[0]:.2f}x + {coef[1]:.2f}") # "r-"
21 plt.legend()
22 plt.show() # affiche la figure à l'écran
23
```

p45ex21.py



```
p48act3.py x
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 def calcul_quantites_avant_et_a_equivalence(i_ajout):
5     v.append(i_ajout) # en nombre d'ajout
6     n_AH.append((0.145*10.0 - 0.100*(VDOSE*i_ajout)) * 0.001) #
7     n_HO.append(0)
8     n_Na.append(0.10 * i_ajout * 0.001)
9     n_A.append(0.10 * i_ajout * 0.001)
10
11 def calcul_quantites_apres_equivalence(i_ajout) :
12     # pass
13     v.append(i_ajout)
14     n_AH.append(0)
15     n_HO.append(0.100*(VDOSE*i_ajout)* 1E-3)
16     n_Na.append(0.100*(VDOSE*i_ajout)* 1E-3)
17     n_A.append(n_A[-1])
18
19 VEQUIVALENCE = 14.5 # Volume équivalence en mL
20 VEAU = 200 # Volume d'eau ajouté en mL
21 VDOSE = 1 # Dose délivrée à chaque ajout du réactif titrant en mL
22 n_AH, n_HO, n_Na, n_A, v = [], [], [], [], [] # initialisation de 5 listes de valeurs
23 for ajout in range(0, 26, 1) : # nb ajout volume titrant
24     if ajout*VDOSE <= VEQUIVALENCE :
25         #appelle une fonction qui calcule les quantités avant et à l'équivalence
26         calcul_quantites_avant_et_a_equivalence(ajout)
27     else :
28         #appelle une fonction qui calcule les quantités après l'équivalence
29         calcul_quantites_apres_equivalence(ajout)
30 # Expression de la conductivité
31 # Eau + V titrant + titree 10mL acide aqueux
32 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)
33
34 # affichage des courbes
35 plt.subplot(121) # subplot permet d'afficher deux graphes
36 plt.plot(v, n_AH, 'bx-', linewidth=0.5, label="n_AH")
37 plt.legend()
38 plt.subplot(122)
39 plt.plot(v, conductance, 'bo', label=r"Conductivité en $\mathrm{mS\cdot m^{-1}}$") # Affichage en Latex
40 plt.legend()
41 plt.show()
42
```

p48act3.py



5.2. Chap 3 méthodes chimiques d'analyse

A Simulation pH-métrique à compléter

```
p66ex20.py x
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from math import log10
4
5
6 CBI = 0.1 # mol.L-1 Concentration initiale en ion benzoate C6H5COO-
7 CA = 0.100 # mol.L-1 Concentration en ion oxonium H3O+ solution titrar
8 V0 = 10 # en mL volume de la solution titrée de benzoate de sodium
9 VE = 10 # en mL volume de solution titrante à l'équivalence
10
11 # Initialisation de 5 tableaux de concentrations en ion benzoate (cb),
12 cb, cbh, ch, ph, v = [], [], [], [], []
13
14 ### Remplissage des 5 listes suivant le volume va ajouté
15 for va in np.arange(1,26,0.5) :
16     if va < VE :
17         v.append(va)
18         cb.append((CBI*V0 - CA*va) / (V0 + va))
19         cbh.append((CA*va / (V0 + va)))
20         ch.append(0)
21         ph.append(4.2 + log10((0.1*V0 - CA*va) / (CA * va)))
22     elif va > VE:
23         pass
24
25 # Affichage des graphiques
26 # Ajuste la taille
27 # Positionner la fenêtre
28 plt.figure(figsize=(10, 5))
29 mngr = plt.get_current_fig_manager()
30 mngr.window.geometry("+300+100")
31 # Dessiner
32 plt.subplot(121)
33 plt.plot(v, cb, 'bx-', label="CB")
34 plt.plot(v, cbh, 'rx-', label="CBH")
35 plt.legend()
36 plt.subplot(122)
37 plt.ylim(0,14)
38 plt.plot(v, ph, 'rx-', label="pH")
39 plt.legend()
40 plt.show()
```

ex20p66.py

```
1 from math import log10
2 import numpy as np
3 import matplotlib.pyplot as plt
4 '''concentration initiale en ions
5 benzoate Cbi et [H3O+] de la solution
6 titrante, volume de la prise d'essai
7 et volume à l'équivalence'''
8 CBi, CA, V0, Ve = 0.1, 0.1, 10, 10
9 '''CB, CBH, CH : liste des concentrations
10 respectives des espèces B-, BH, H3O+ et
11 du pH et du volume'''
12 CB, CBH, CH, pH, V = [], [], [], [], []
13 for VA in np.arange(1,26,0.5) :
14     if VA < Ve :
15         V.append(VA)
16         CB.append((CBI*V0 - CA*VA) / (V0 + VA))
17         CBH.append((CA*VA / (V0 + VA)))
18         CH.append(0)
19         pH.append(4.2 + log10((0.1*V0 - CA*VA) \
20 / (CA*VA)))
21     elif VA > Ve :
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489

```

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

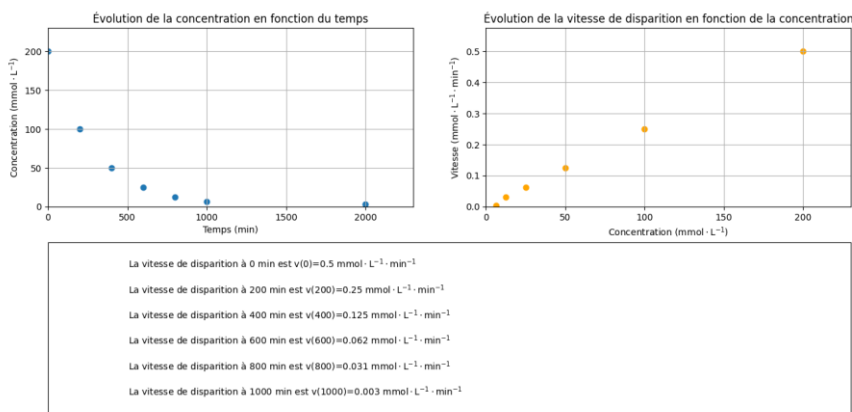
p89ex20.py ×

```
1 import matplotlib.pyplot as plt
2 # Compléter les listes suivantes à partir de l'exercice 13
3 t = [0, 200, 400, 600, 800, 1000, 2000] # date en min
4 c = [200, 100, 50, 25, 12.5, 6.3, 3.1] # concentration en mmol.L-1
5 # calcule des vitesses de disparition
6 vd = []
7 # pour indice de 1er date à avant dernière calculer vd
8 for i in range(len(t)-1):
9     vd = vd + [-1*((c[i+1]-c[i]) / (t[i+1]-t[i]))] # On veut une vitesse positive
10 # recherche des valeurs max pour dimensionner les graphiques
11 c_max = max(c)
12 t_max = max(t)
13 vd_max = max(vd)
14 # On veut disposer d'un deuxième jeu d'abscisse pour le graphique de vitesse vd = f(concentration)
15 c_2 = list(c) # On fait une copie (superficielle)
16 del c_2[-1] # -1 est l'indice du dernier élément
17 c2_max = max(c_2)
18 plt.figure(figsize=(16,8))
19 #sous-figure 1 - Graphique donnant l'évolution de la concentration en fonction du temps
20 plt.subplot(221) # 2 lignes 2 colonnes 1er cadre
21 plt.grid()
22 plt.title("Évolution de la concentration en fonction du temps")
23 plt.xlim(0, 1.15*t_max) # Pour avoir un graphe plus grand de 15%
24 plt.ylim(0, 1.15*c_max)
25 plt.scatter(t, c) # permet d'afficher un nuage de points, la fonction plt.plot les affiche reliés par défaut
26 plt.xlabel("Temps (min)")
27 plt.ylabel(r"Concentration ($\mathrm{mmol} \cdot \mathrm{L}^{-1}$)") # Latex
28 #sous-figure 2- Graphique donnant l'évolution de la vitesse de disparition en fonction du temps
29 plt.subplot(222) # 2 lignes 2 colonnes 2ieme cadre
30 plt.grid()
31 plt.title("Évolution de la vitesse de disparition en fonction de la concentration")
32 plt.xlim(0,1.15*c2_max)
33 plt.ylim(0,1.15*vd_max)
34 plt.scatter(c_2, vd, color='orange')
35 plt.ylabel(r"Vitesse ($\mathrm{mmol} \cdot \mathrm{L}^{-1} \cdot \mathrm{min}^{-1}$)")
36 plt.xlabel(r"Concentration ($\mathrm{mmol} \cdot \mathrm{L}^{-1}$)")
37 #afficher dans la fenêtre (On fait un graphique vide)
38 plt.subplot(212, frameon=True) # 2 ligne 1 colonne cadre ligne 2
39 plt.xticks([]) # [n] graduation à indiquer ici aucune
40 plt.yticks([])
41 for i in range(len(t)-1):
42     plt.text(0.1,
43             ((len(t)-1)-i)/len(t),
44             "La vitesse de disparition à " +
45             str(t[i]) + " min est v(" + str(t[i]) +
46             ")=" +
47             str(round(vd[i],3)) +
48             r"$\mathrm{mmol} \cdot \mathrm{L}^{-1} \cdot \mathrm{min}^{-1}$",
49             fontsize = 10 )
50 plt.show()
```

A Extract from the Python program

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 '''compléter les listes suivantes et
4 les décommenter'''
4 # t = [...]
5 # C = [...]
```

p89ex20.py

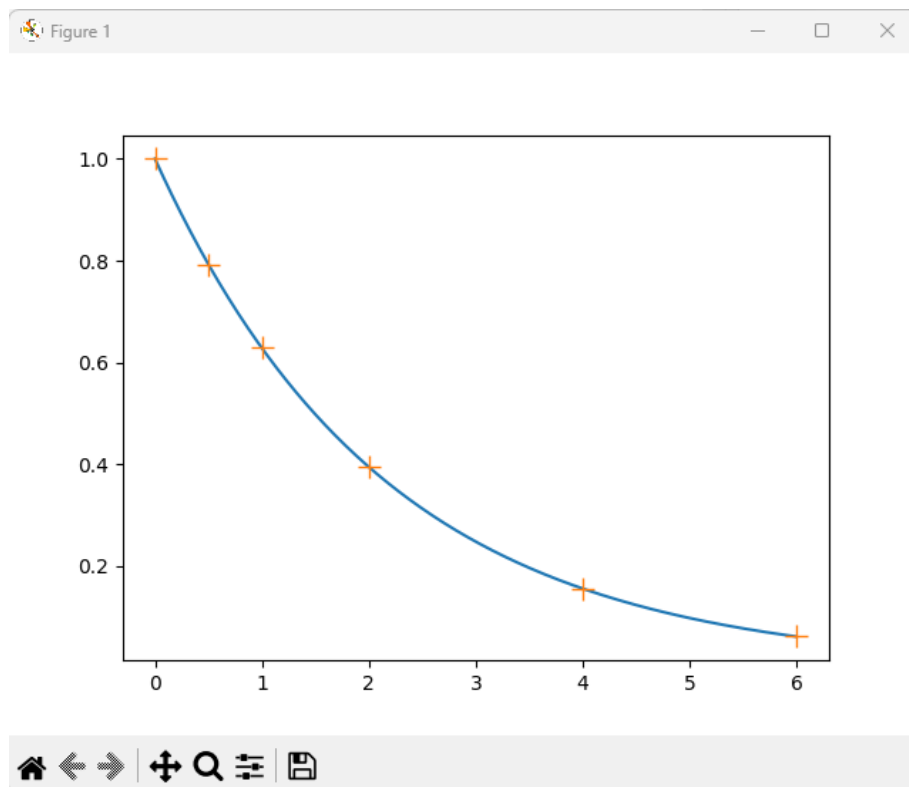


p90ex23.py ×

```
1 import matplotlib.pyplot as plt
2 # ecart entre deux dates de mesure
3 DELTA_T = 0.01
4 # Nombre de point de mesure
5 N = 600
6 # initialiser un tableau de N dates
7 t = [i*DELTA_T for i in range(N)]
8 # Initialiser une tableau de concentration nulles
9 c = [0] * N
10 # Définir la première concentration
11 c[0] = 1.000 # en mol.L-1
12 K = 0.464 # Coef en h-1
13 # Pour indice date de e la première à l'avant dernière
14 for i in range(N-1):
15     c[i+1] = c[i] - (t[i+1] - t[i]) * K * c[i]
16 # On prepare le tracé
17 plt.plot(t, c)
18 # On prépare les mesure réelles
19 t_mes = [0, 0.5, 1, 2, 4, 6]
20 c_mes = [1.000, 0.793, 0.630, 0.396, 0.155, 0.063]
21 # On prépare la nouvelle courbe
22 plt.plot(t_mes, c_mes, '+', markersize=12) # pour agrandir le +
23 # On trace
24 plt.show()
```

```
1 import matplotlib.pyplot as plt
2 Delta_t=0.01
3 # Delta_t est considéré comme petit
4 N=600
5 t=[i*Delta_t for i in range(N)]
6 C=[0]*N
7 # Données de l'énoncé :
8 C[0]=1.000
9 k=0.464
10 for i in range(N-1):
11     C[i+1]=C[i]-(t[i+1]-t[i])*k*C[i]
12 plt.plot(t,C)
13 t_mes=[0, 0.5, 1, 2, 4, 6]
14 C_mes=[1.000, 0.793, 0.630, 0.396, 0.155,\
15 0.063]
15 plt.plot(t_mes,C_mes,'+',markersize=12)
```

p90ex23.py



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

```
p94act1.py x
1 import matplotlib.pyplot as plt
2 from random import randint, random
3
4 def creation_echantillon(na, nb, nc, nd) :
5     '''
6     Renvoie un echantillon de na espèce A, n b espèce B, nc espèce C, nd espèce D
7     na, nb, nc, nd : int par exemple 1, 2, 3, 4
8     echantillon [str] par exemple ['A', 'B', 'B', 'C', 'C', 'C', 'D', 'D', 'D', 'D']
9     '''
10    echantillon = []
11    for _ind in range(na) :
12        echantillon.append("A")
13    for _ind in range(nb) :
14        echantillon.append("B")
15    for _ind in range(nc) :
16        echantillon.append("C")
17    for _ind in range(nd) :
18        echantillon.append("D")
19    return echantillon
20
21 def comptage_especes(ech, nb_a, nb_b, nb_c, nb_d): # fonction qui compte le nombre d'espèces
22     '''
23     Dénombre dans l'echantillon ech le nombre de chaque espèce A, B, C, D
24     Ajoute en place ces nombres dans les listes nb_a, nb_b, nb_c, nb_d
25     '''
26    n_entite = len(ech)
27    nbea, nbeb, nbec, nbed = 0,0,0,0
28    for j in range(n_entite):
29        if ech[j] == "A" :
30            nbea = nbea + 1
31        elif ech[j] == "B" :
32            nbeb = nbeb + 1
33        elif ech[j] == "C" :
34            nbec = nbec + 1
35        elif ech[j] == "D" :
36            nbed = nbed + 1
37    nb_a.append(nbea)
38    nb_b.append(nbeb)
39    nb_c.append(nbec)
40    nb_d.append(nbed)
41
42 def collision(pac, collision_ab) : # fonction qui génère des collisions et remplace A, B éventuellement
43    n_entite = len(pac) # nombre d'entité dans le sac
44    # définir deux indices différents
45    n1 = randint(0, n_entite-1)
46    n2 = randint(0, n_entite-1)
47    while n2 == n1 :
48        n2 = randint(0, n_entite-1)
49    # si c'est un choc AB
50    if (pac[n1] == "A" and pac[n2] == "B") or (pac[n1] == "B" and pac[n2] == "A") :
51        x = random() # au hasard entre
52        if x > collision_ab :
53            pac[n1], pac[n2] = "C", "D"
54    return pac
```

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

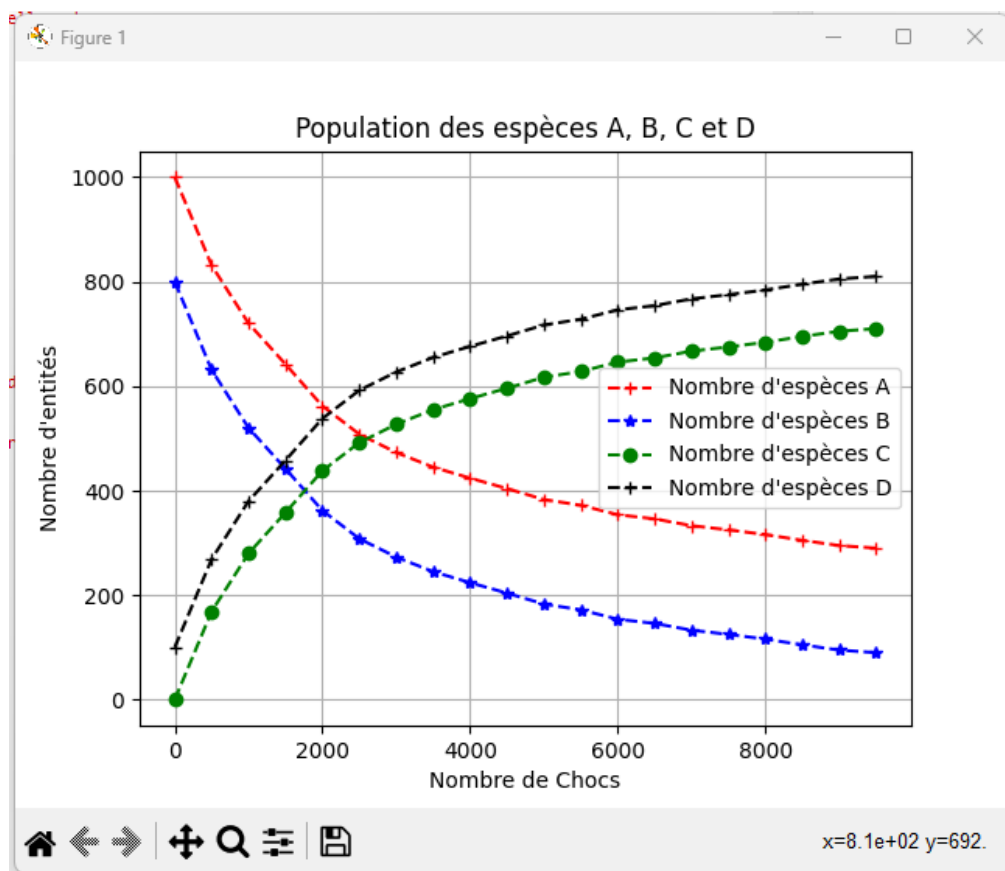


```

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'événements
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



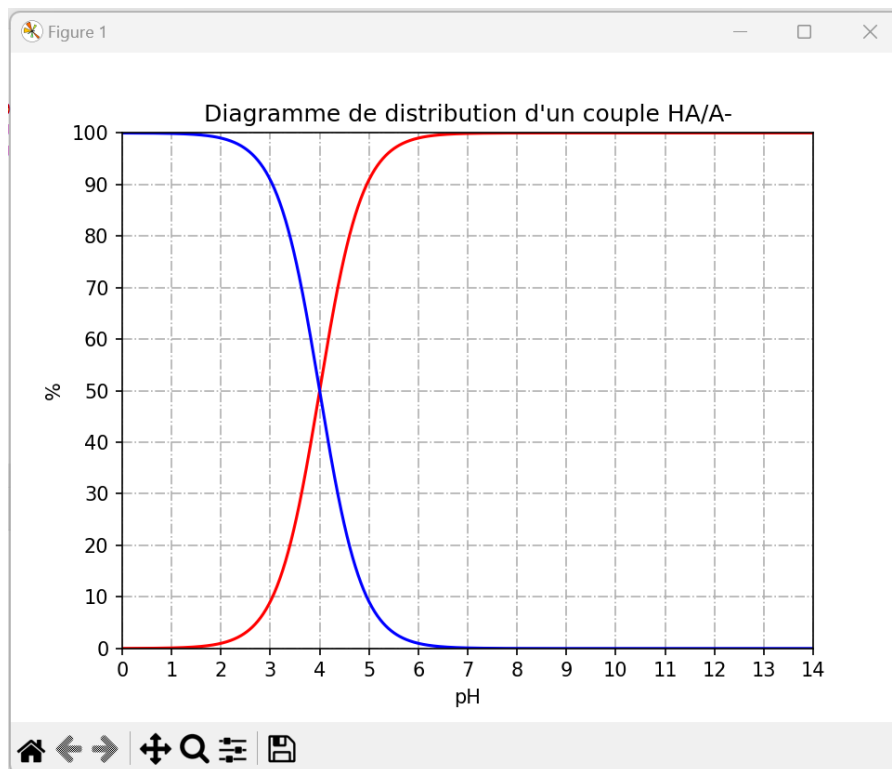
5.5. Chap 8 Force des acides et des bases

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_ad, color='r', label='% en HA' )
15 plt.plot(ph, ph_ac, 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 pH_A = [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, pH_A, 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
```

p167ex24.py

```
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')
```

Rendu dans la console :

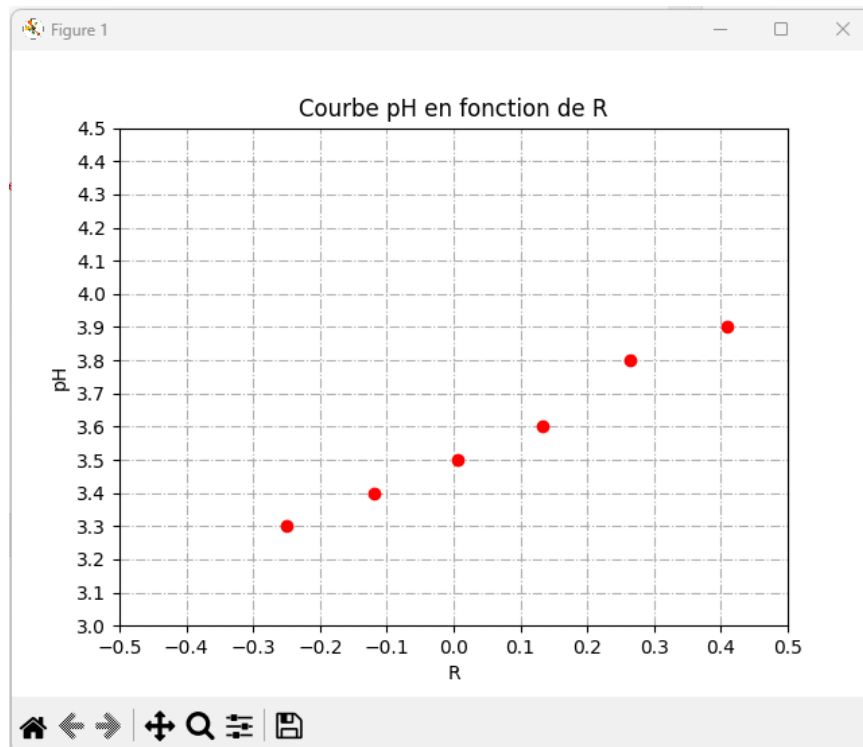
```
>>> %Run p167ex24.py
pH de la solution ? pH = 1.3
Concentration en soluté apporté en mol.L-1 ? C = 10.5
Volume de la solution en L ? V = 0.100
tau d'avancement = 0.71 Acide faible
>>>
```

p171ece.py ×

```
1 import matplotlib.pyplot as plt
2 from numpy import log10, arange
3
4 # Concentration en hydroxyde de sodium dans la burette
5 CB = 0.10 # en mol.L-1
6 # Masse d'acide acétylsalicylique
7 MA = 500 # en mg
8 # Masse molaire d'acétylsalicylique
9 M = 180 #en g.mol-1
10 # Mesures
11 v_b = [0.010, 0.012, 0.014, 0.016, 0.018, 0.020] # en L volume ajoutés par burette
12 ph = [3.3, 3.4, 3.5, 3.6, 3.8, 3.9] # pH de la solution
13 # calculs des coefficients r
14 r = [] # coefficient R
15 for v in v_b :
16     r.append(log10(CB*v / (MA*1E-3/M - CB*v)))
17 # tracé
18 plt.title('Courbe pH en fonction de R')
19 plt.xlabel('R')
20 plt.ylabel('pH')
21 plt.axis(xmin=-0.5, xmax=0.5, ymin=3, ymax=4.5)
22 plt.xticks(arange(-0.5, 0.6, 0.1)) # 1 graduation
23 plt.yticks(arange(3, 4.6, 0.1))
24 plt.grid(linestyle='-.')
25 plt.plot(r, ph, 'ro')
26 plt.show()
```

```
1 import matplotlib.pyplot as plt
2 from numpy import log10, arange
3 Vb = [ ]
4 pH = [ ]
5 R=[]
6 for i in Vb :
7     R.append(log10(0.1*i/(0.5/180-0.1*i)))
8 plt.title('Courbe pH en fonction de R')
9 plt.xlabel('R')
10 plt.ylabel('pH')
11 plt.axis(xmin=-0.5, xmax=0.5, ymin=3, ymax=4.5)
12 plt.xticks(arange(-0.5,0.6,0.1))
13 plt.yticks(arange(3,4.6,0.1))
14 plt.grid(linestyle='-.')
15 plt.plot(R, pH, 'ro')
16 plt.show()
```

p171ece.py



5.6. Chap 11 Mouvement et deuxième loi de Newton

p232ex24.py ×

```

1 import csv
2 import matplotlib.pyplot as plt
3
4 def import_csv (cible, sep, n):
5     """
6     Renvoyer les données d'une colonne d'un fichier csv de séparateur choisi.
7     cible : str chemin absolu ou relatif du fichier"
8     sep : str le séparateur utilisé (; ou ,)
9     n : int la colonne souhaitée ( à partir de 0)
10    rep : liste [ [float]...] les valeurs numériques de la colonne sélectionnée
11    """
12    rep = []
13    with open(cible, 'r', encoding='utf-8', newline='') as fichier:
14        fichier.readline() # elimine la première ligne des noms des champs si nécessaire
15        reader = csv.reader(fichier, delimiter=sep)
16        for row in reader:
17            notation_point = row[n].replace(",",".") # changer le format des nombres "3,5" "3.5"
18            rep.append(float(notation_point)) # Transformer en nombre les str
19    return rep
20
21 K = 0.02 # Facteur echelle pour les accélérations
22 # Nom du fichier à traiter son chemin csv\balle_de_tennis $attention \ pour les chemins et non /
23 # m_fichier = input("Quel est le nom du fichier de pointage (sans l'extension .csv)? ") + ".csv"
24 m_fichier = r"csv\balle_de_tennis.csv"
25 # Lire les données.
26 t = import_csv(m_fichier, ";", 0) # la date en s
27 x = import_csv(m_fichier, ";", 1) # La position abscisse en m
28 y = import_csv(m_fichier, ";", 2) # La position ordonnée en m
29

```

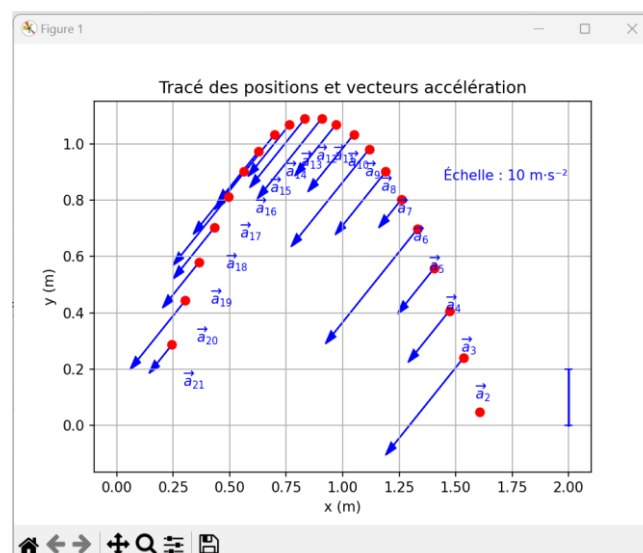
```

35 """Calcul des coordonnées vx et vy des
    vecteurs vitesse"""
36 vx=[ ]
37 for i in range(len(x)-1) :
38     vx=vx+[(x[i+1]-x[i])/(t[i+1]-t[i])]
39
40 vy=[ ]
41 for i in range(len(y)-1) :
42     vy=vy+[(y[i+1]-y[i])/(t[i+1]-t[i])]
43
44 """Calcul des coordonnées ax et ay des
    vecteurs accélération"""
45 ax=[ ]
46 for i in range(len(vx)-1) :
47     ax=
48
49 ay=[ ]
50 for i in range(len(vy)-1) :
51     ay=

```

```
p232ex24.py ×
30 # Calcul des coordonnées vx et vy des vecteurs vitesse
31 vx = []
32 for i in range(len(x)-1) :
33     vx = vx + [(x[i+1]-x[i]) / (t[i+1]-t[i])]
34 vy = []
35 for i in range(len(y)-1) :
36     vy = vy + [(y[i+1]-y[i]) / (t[i+1]-t[i])]
37 # Calcul des coordonnées ax et ay des vecteurs accélération
38 ax = []
39 for i in range(len(vx)-1) :
40     ax = ax + [(vx[i+1]-vx[i]) / (t[i+1]-t[i])]
41 ay = []
42 for i in range(len(vy)-1) :
43     ay = ay + [(vy[i+1]-vy[i]) / (t[i+1]-t[i])]
44
45 # Légende
46 plt.title("Tracé des positions et vecteurs accélération")
47 plt.xlabel('x (m)')
48 plt.ylabel('y (m)')
49 plt.grid()
50 plt.text(0.9*max(x), 0.8*max(y), f"Échelle : {10} m·s-2", color="blue")
51 plt.plot([2, 2],[0, 10*K], '._-', color='blue') # trace un trait d'échelle
52 # Tracé
53 for i in range(len(t)-1):
54     plt.plot(0, 0, x[i], y[i], 'ro') # position
55     # vecteurs accélération
56     if i < len(t)-2:
57         plt.arrow(x[i+1], y[i+1],
58                   ax[i]*K, ay[i]*K, #appliquer le facteur echelle
59                   color='b',
60                   head_width=0.03,
61                   head_length=0.05,
62                   length_includes_head=True)
63         plt.text(x[i+1]+0.05, y[i+1]-0.15,
64                  r'$\overrightarrow{a}_{\%.i}$'%(i+2),
65                  color="blue")
66 plt.show()
67 #sauvegarde sous forme d'image
68 plt.savefig("vecteurs-accélération.png")
```

p232ex24



5.7. Chap 12 Mouvement et interaction dans un champ uniforme

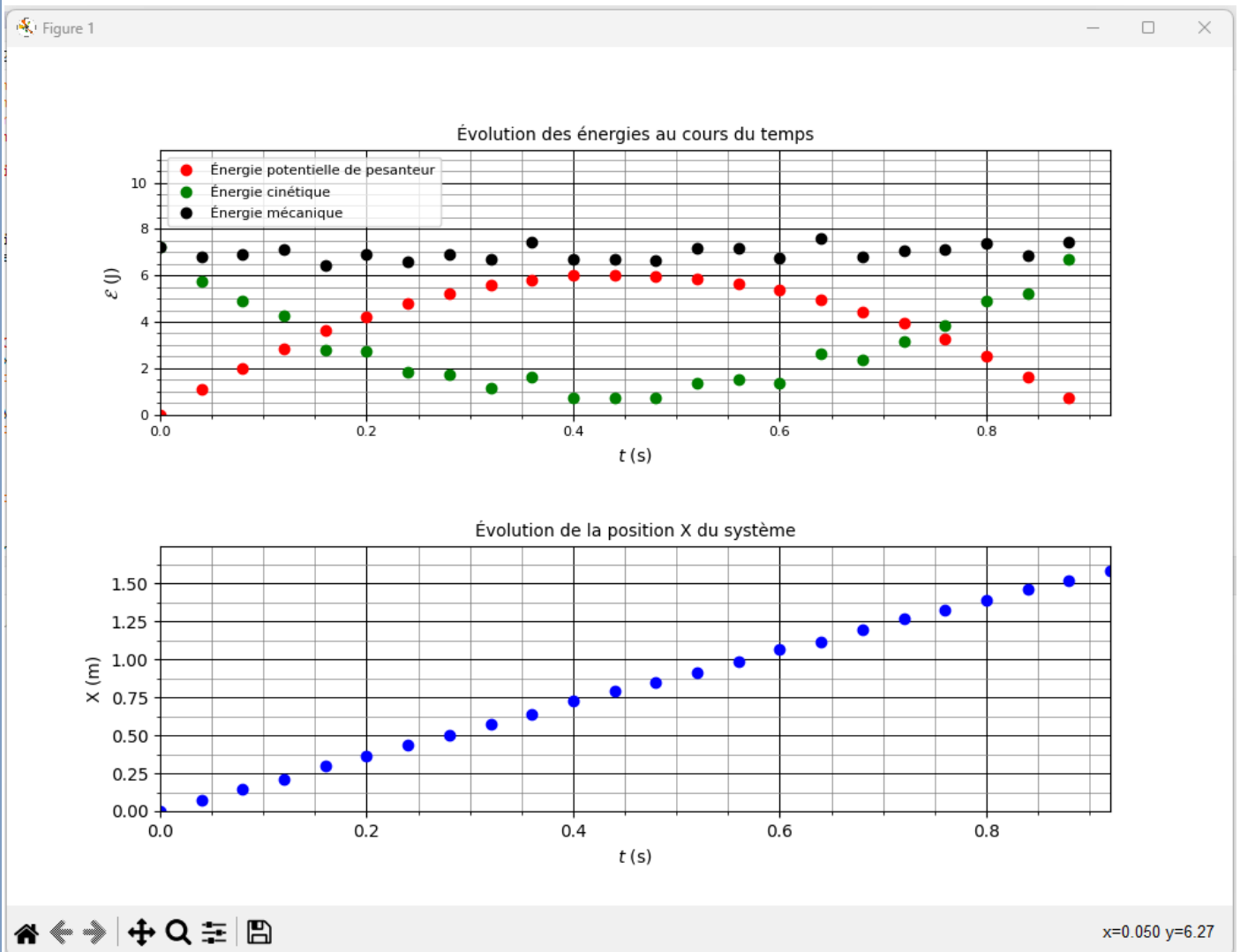
```
p240act2.py x
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 from matplotlib.ticker import AutoMinorLocator
4 #masse du système
5 M = 0.60 # En kg
6 #intensité de la pesanteur g
7 G = 9.81 # en N.kg-1
8 # Lecture du fichier
9 # fichier = input("Quel est le nom du fichier de pointage (sans l'extension .csv)?")+ ".csv"
10 fichier = "csv\\mouvement_parabolique.csv"
11 data = pd.read_csv(fichier, sep=';', header=0) # suppose séparateur (;ou,) et nom des champs connus !
12 # typage (, ->.) et str-> float
13 t = list(pd.to_numeric(data['Temps'].str.replace(',', '.')).values())
14 x = list(pd.to_numeric(data['X'].str.replace(',', '.')).values())
15 y = list(pd.to_numeric(data['Y'].str.replace(',', '.')).values())
16 #Calcul des coordonnées Vx et Vy des vecteurs vitesse
17 vx=[]
18 for i in range(len(x)-1):
19     vx = vx + [(x[i+1]-x[i]) / (t[i+1]-t[i])]
20 vy=[]
21 for i in range(len(y)-1):
22     vy = vy + [(y[i+1]-y[i]) / (t[i+1]-t[i])]
23 # calcule de la vitesse
24 v = []
25 for i in range(len(vx)):
26     v = v + [(vx[i]**2 + vy[i]**2)**0.5] # racine carrée ou puissance 1/2
27 # calcul des énergies
28 energie_p = [] # Energie potentielle
29 for i in range(len(vx)):
30     energie_p = energie_p + [M*G*y[i]] # y est l'axe vertical orienté vers le haut
31 energie_c = [] # Energie cinétique
32 for i in range(len(vx)):
33     energie_c = energie_c + [0.5*M*v[i]**2]
34 energie_m = [] # Energie mécanique
35 for i in range(len(vx)):
36     energie_m = energie_m + [energie_c[i] + energie_p[i]]
37 # Préparation du graphique
38 # Ajuster la taille et la position de la fenêtre
39 plt.figure(figsize=(10, 7))
40 mngr = plt.get_current_fig_manager()
41 mngr.window.geometry("+50+50")
42 # Ajustement du plot
43 plt.subplots_adjust(hspace=0.5)
44 # Sous figure supérieure
45 plt.subplot(211)
46 plt.ylim([0, 1.5*max(energie_m)])
47 plt.xlim([0, max(t)])
48 #Parametres de la grille
49 axes = plt.gca()
50 axes.minorticks_on()
51 axes.grid(which='major', linestyle='-', linewidth='0.7', color='black')
52 axes.grid(which='minor', linestyle='-', linewidth='0.5', color='grey')
53 # Attention le temps dispose d'une valeur de plus !
54 # Tracé des énergies
55 plt.plot(t[0:-1], energie_p, 'ro', label='Énergie potentielle de pesanteur')
56 plt.plot(t[0:-1], energie_c, 'go', label='Énergie cinétique')
57 plt.plot(t[0:-1], energie_m, 'ko', label='Énergie mécanique')
58 #Légendes
59 plt.title("Évolution des énergies au cours du temps", fontsize=10)
60 plt.ylabel(r'$\mathcal{E}$ (J)', fontsize=10)
61 plt.xlabel('t$ (s)$', fontsize=10)
62 plt.tick_params(labelsize=8)
63 plt.legend(loc=2, fontsize="8")
64 # Sous figure inférieure
65 plt.subplot(212)
66 plt.ylim([0, 1.1*max(x)])
67 plt.xlim([0, max(t)])
68 # Parametres de la grille
69 axes2 = plt.gca()
70 axes2.minorticks_on()
71 axes2.grid(which='major', linestyle='-', linewidth='0.7', color='black')
72 axes2.grid(which='minor', linestyle='-', linewidth='0.5', color='grey')
73 axes2.yaxis.set_minor_locator(AutoMinorLocator(2))
74 #tracé de la position
75 plt.plot(t, x, 'bo')
76 #Légendes
77 plt.title("Évolution de la position X du système", fontsize=10)
78 plt.ylabel('X (m)', fontsize=10)
79 plt.xlabel('t$ (s)$', fontsize=10)
80 plt.tick_params(labelsize=10)
81 # tracé
82 plt.show()
```

p240act2.py


```

30 v=[]
31 for i in range(len(vx)) :
32     v=v+[(vx[i]**2+vy[i]**2)**0.5]
33 #masse du système en kg
34 m=0.60
35 #intensité de la pesanteur
36 g=9.81
37 #Calcul des énergies
38 Ep=[]
39 for i in range(len(vx)) :
40     Ep=Ep+[m*g*y[i]]
41 Ec=[]
42 for i in range(len(vx)) :
43     Ec=
44 Em=[]
45 for i in range(len(vx)) :
46     Em=Em+[Ep[i]+Ec[i]]
47 plt.show()

```



5.8. Chap 13 Mouvement dans un champ de gravitation

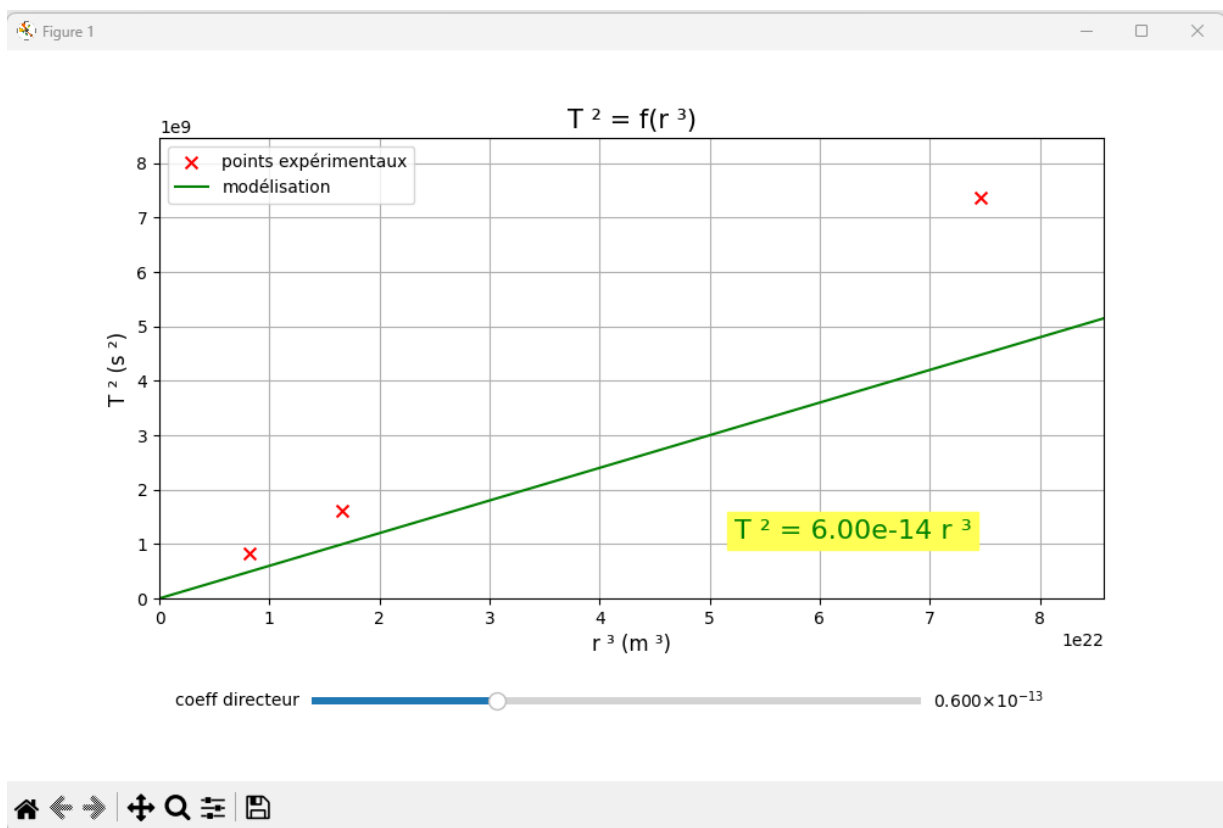
```
p273ex18.py ×
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4 from matplotlib.widgets import Slider
5
6 def update(val):
7     ...
8     Actualiser la droite de modélisation et son équation,
9     val float la valeur du curseur
10    modifie en place
11    periode_ma (numpy.ndarray)
12    init_a[0] float
13    ...
14    periode_aj = val * rayon_ma # Nouvelle equation y = ax
15    modele[0].set_ydata(periode_aj)
16    # le tracé est mis à jour en arrière plan
17    # il faut mettre à jour l'objet equation
18    equation.set_text(f'T² = {val:.2e} r³')
19
20 # Obtention des données type csv
21 fichier = 'csv\\Satellites_artificiels.csv'
22 data = pd.read_csv(fichier, sep=';', header=0)
23 # Mise en forme des données, periode et rayon sont des objets pandas !
24 periode = data['T(s)']**2 # On conserve T²
25 rayon = (data['a(km)']*1E3)**3 # On conserve r³
26 # Calcul de la régression linéaire f sur T² = f(r³)
27 regression = np.polyfit(rayon, periode, 1) # renvoie les coefficients a, b d'un modèle ax+b
28 pente_moyenne = regression[0]
29 # initialisation d'un modèle ajustable y = ax
30 init_a = 6.00e-14 #pente initiale du modèle
31 rayon_ma = np.linspace(0, rayon[2]*2, 5) # rayon du modele ajustable
32 periode_ma = init_a * rayon_ma # periode du modele ajustable
33
34 # Tracé des points expérimentaux et de la régression linéaire
35 # taille et position de la fenêtre
36 plt.figure(figsize=(10,6))
37 plt.get_current_fig_manager().window.geometry("+5+5")
38 # Ajustement du plot
39 plt.subplots_adjust(bottom=0.25) #on laisse un espace pour le curseur
40 # définition des axes
41 plt.axis([0, max(rayon)*1.15, 0, max(periode)*1.15])
42 plt.grid() # Paramètre de la grille
43 # légendes des axes
44 plt.xlabel('r³ (m³)', fontsize=12)
45 plt.ylabel('T² (s²)', fontsize=12)
46 plt.tick_params(labelsize=10)
47 plt.title('T² = f(r³)', fontsize=15)
48 # Tracé des points de mesure
49 plt.scatter(rayon, periode, s=50, color='r', marker='x', label='points expérimentaux')
50 # Affichage du modèle ajustable, modele[0] de classe matplotlib.lines.Line2D
51 modele = plt.plot(rayon_ma, periode_ma, '-g', label='modélisation')
52 plt.legend(loc=2, fontsize=10) #Légende du graphique
53 # affichage de l'équation dans le graphique
54 equation = plt.text(max(rayon)*0.7,max(periode)*0.15,
55                     f'T² = {init_a:.2e} r³',
56                     fontsize = 16, color="green", backgroundcolor = "#FFFF55")
57 # Création d'un curseur
58 rectangle_a = plt.axes([0.25, 0.1, 0.5, 0.02])
59 curseur = Slider(rectangle_a, 'coeff directeur', 0, pente_moyenne*2, valinit=init_a)
60 # appel update lorsque le curseur est modifié
61 curseur.on_changed(update)
62 plt.show() # tracé en boucle
```

p273ex18.py

```

28 #Variables et régression linéaire
29 #Calcul des variables associées à la fonction
30 Tcarre = T**2
31 rcube = (r*1000)**3
32 #Calcul de la régression linéaire
33 regression = np.polyfit(rcube, Tcarre,1)
34 pente_moyenne=regression[0]
35 init_a=6.00e-14
36 #Variables pour la régression
37 rcube_regression=np.linspace(0,rcube[2]*2,5)
38 T_regression = init_a*rcube_regression

```



Numérique et sciences informatiques

p297syn.py

```
19 vx=[]
20 for i in range(len(x)-1):
21     vx=vx+[(x[i+1]-x[i])/(t[i+1]-t[i])]
22
23 vy=[]
24 for i in range(len(y)-1):
25     vy=vy+[(y[i+1]-y[i])/(t[i+1]-t[i])]
```

Luc.vincent@ac-bordeaux.fr

5.10. Chap 18 diffraction et interférence

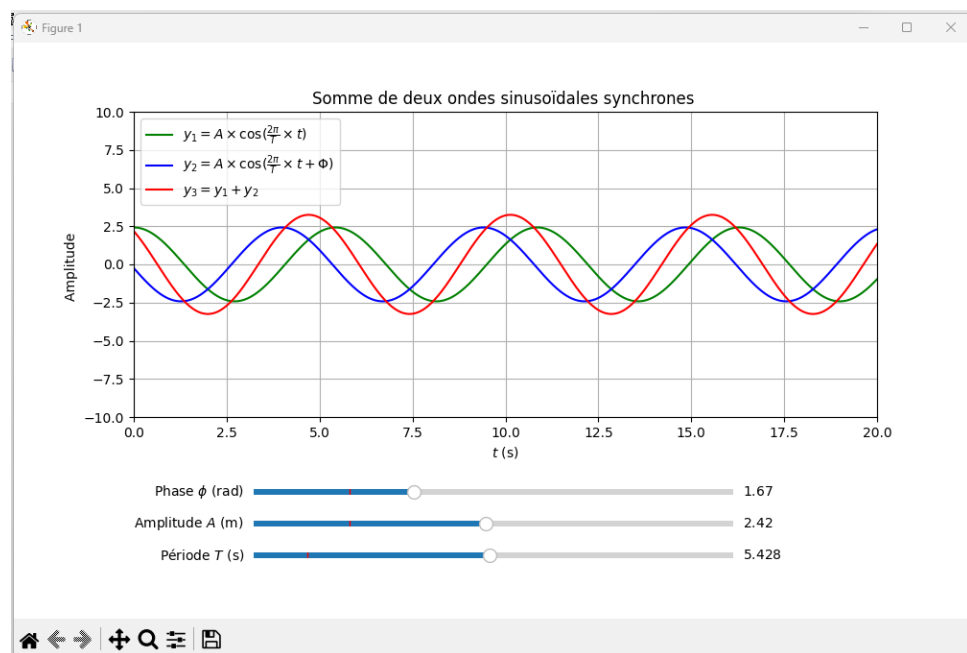
```
p381ex22.py x
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from matplotlib.widgets import Slider
4
5 #Fonction d'actualisation des courbes
6 def update(_x):
7     ...
8     Actualiser le graphique
9     x float la valeur de l'un des curseurs
10    variables globales curseur_A, curseur_T, curseur_PHI
11    Modifie en place les tracés p1, p2 et p3
12    ...
13    # Nouveaux coefficients des objets curseurs
14    A = curseur_A.val
15    T = curseur_T.val
16    PHI = curseur_PHI.val
17    # Calcul nouveau jeux de valeurs
18    y_1 = A * np.cos(2*np.pi/T * time)
19    y_2 = A * np.cos(2*np.pi/T * time + PHI)
20    y_3 = y_1 + y_2
21    #mis à jour des tracés
22    p1[0].set_ydata(y_1)
23    p2[0].set_ydata(y_2)
24    p3[0].set_ydata(y_3)
25
26
27 # valeurs par défaut
28 AMP = 1.00 # Amplitude
29 PER = 2.00 # Période
30 DPHA = 1.00 # Déphasage
31 # Définitions des courbes
32 time = np.linspace(0, 20, 2000)
33 y1 = AMP * np.cos(2*np.pi/PER * time)
34 y2 = AMP * np.cos(2*np.pi/PER * time + DPHA)
35 y3 = y1 + y2
36
37 #Tracé des courbes
38 # taille et position de la fenêtre
39 plt.figure(figsize=(10,6))
40 plt.get_current_fig_manager().window.geometry("+5+5")
41 # Ajustement du plot
42 plt.subplots_adjust(bottom=0.35) #on laisse un espace pour le curseur
43 # définition des axes
44 plt.axis([0, 20, -10, 10])
45 plt.grid() # Paramètre de la grille
46 # légendes des axes
47 plt.xlabel('$t$ (s)')
48 plt.ylabel('Amplitude ')
49 plt.title('Somme de deux ondes sinusoïdales synchrones ')
50
51 # tracé des courbes pn[0] de classe matplotlib.lines.Line2D
52 p1 = plt.plot(time, y1, '-g',label=r'$y_1 = A \times \cos(\frac{2\pi}{T} \times t)$')
53 p2 = plt.plot(time, y2, '-b',label=r'$y_2 = A \times \cos(\frac{2\pi}{T} \times t + \Phi)$')
54 p3 = plt.plot(time, y3, '-r',label=r'$y_3 = y_1 + y_2$')
55 plt.legend(loc=2, fontsize=10) #Légende du graphique
56
57
58 # Création des curseurs
59 # Curseur periode
60 rectangle_a = plt.axes([0.25, 0.1, 0.5, 0.02])
61 curseur_T = Slider(rectangle_a, 'Période $T$ (s)', 1, 10, valinit=PER)
62 # Curseur amplitude
63 rectangle_b = plt.axes([0.25, 0.155, 0.5, 0.02])
64 curseur_A = Slider(rectangle_b, 'Amplitude $A$ (m)', 0, 5, valinit=AMP)
65 # Curseur phase
66 rectangle_c = plt.axes([0.25, 0.210, 0.5, 0.02])
67 curseur_PHI = Slider(rectangle_c, r'Phase $\phi$ (rad)', 0, 5, valinit=DPHA)
68
69 # appel update lorsque l'un des curseurs est modifié
70 curseur_T.on_changed(update)
71 curseur_A.on_changed(update)
72 curseur_PHI.on_changed(update)
73
74 plt.show()
```

p381ex22.py

```

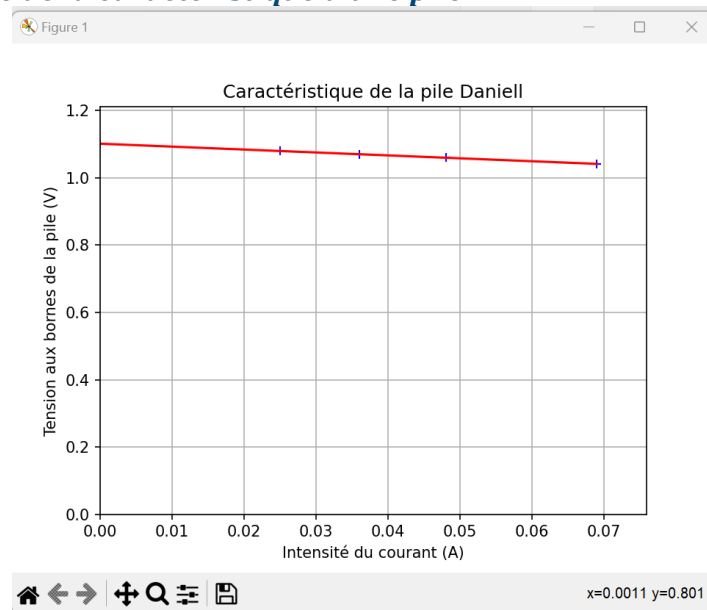
19 #Conditions initiales
20 initial_A = 1.00
21 initial_T = 2.00
22 initial_Phi = 1.00
23 # Définitions des courbes
24 time=np.linspace(0.,20.,2000)
25 y1=initial_A*np.cos(2*np.pi/initial_T*time)
26 y2=initial_A*np.cos(2*np.pi/initial_T*\
    time+initial_Phi)
27 y3=y1+y2
28 #Tracé des courbes
29 G = GridSpec(10, 10)
30 fig, ax = plt.subplots()
31 axes_1 = plt.subplot(G[:-3, :])
32 plt.axis([0,20,-10,10])
33 plt.xlabel('t en s')
34 plt.ylabel('Amplitude')
35 plt.title('Somme de deux ondes sinusoïdales \
    synchrones')
36 plt.grid()
37 p1, = plt.plot(time, y1, '-g', label=\
    r'$y_1=A\cos(\frac{2\pi}{T}t)$')
38 p2, = plt.plot(time, y2, '-b', label=\
    r'$y_2=A\cos(\frac{2\pi}{T}t+\Phi)$')
39 p3, = plt.plot(time, y3, '-r',label=\
    r'$y_3= y_1 + y_2$')
40 plt.legend()

```



6. Exercices

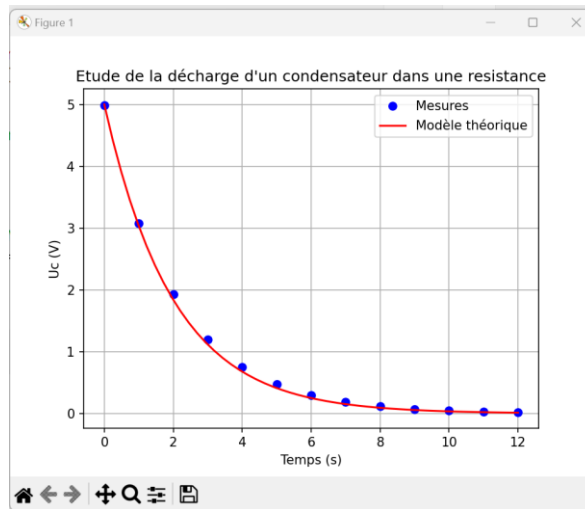
6.1. Tracé de la caractéristique d'une pile



```
ex6_1.py x
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Points expérimentaux
5 intensite = [0, 0.025, 0.036, 0.048, 0.069] # en A
6 tension = [1.1, 1.08, 1.07, 1.06, 1.04] # en V
7
8 # A compléter
```

6.2. Tracé de la décharge d'un condensateur

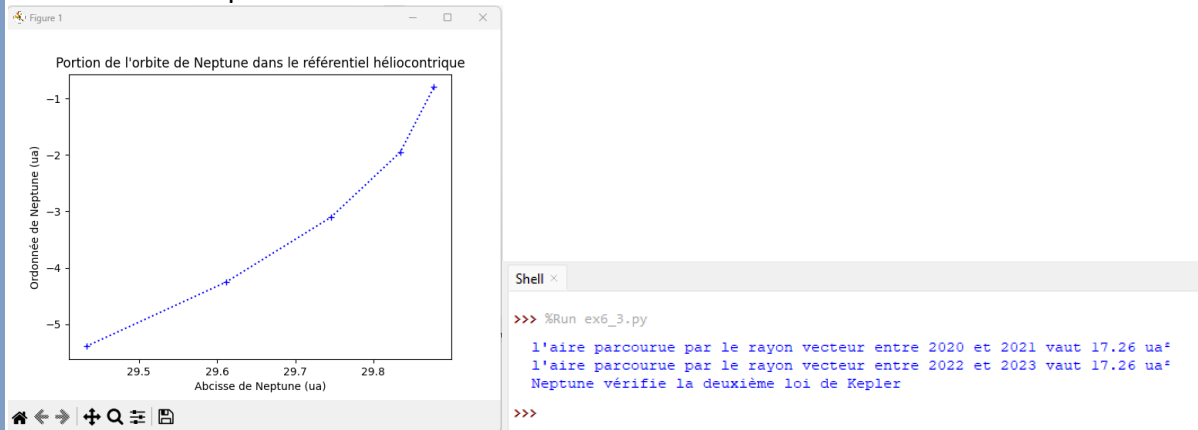
On veut obtenir un graphique à partir de mesures, puis placer sur ces points un modèle mathématique connu.



```
ex6_2.py
1 # Tracé des courbes théoriques et expérimentales
2 # de la décharge d'un condensateur de 100 µF
3 # dans un conducteur ohmique de 20kΩ
4
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 C = 100E-6 # en farad
9 R = 20E3 # en ohm
10 UINIT = 5 #en volt
11
12 # valeurs expérimentales
13 # date en seconde
14 t_ex = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
15                 10, 11, 12])
16 # tensions aux bornes du condensateur en volt
17 uc_ex = np.array([4.99, 3.08, 1.93, 1.2, 0.75,
18                  0.47, 0.3, 0.19, 0.12, 0.07, 0.05,
19                  0.03, 0.02])
20 # A compléter
```

6.3. Vérification de la deuxième loi de Kepler

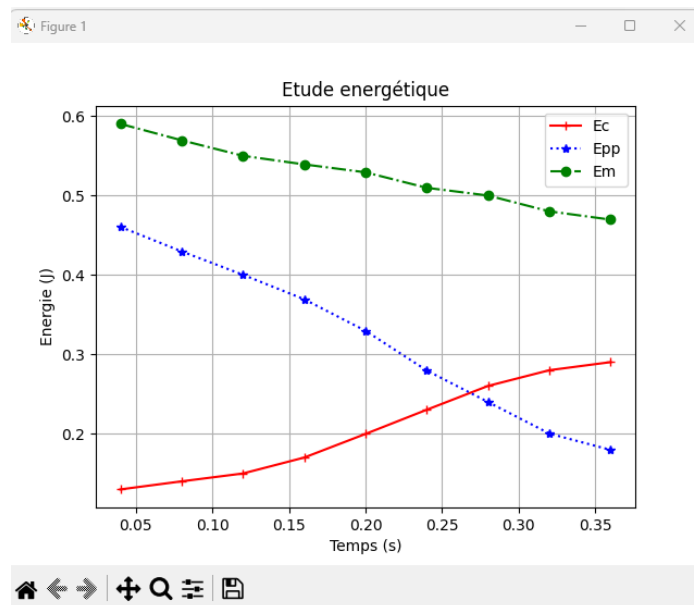
On veut vérifier que des mesures vérifient une loi connue.



```
ex6_3.py
1 # Vérification de la deuxième loi de Kepler
2 import matplotlib.pyplot as plt
3 import math as m
4
5 # valeurs expérimentales
6 # date en année
7 t_ex = [2020, 2021, 2022, 2023, 2024]
8 # Coordonnées en unité astronomique ua
9 x = [29.4328410560730, 29.6118356847488,
10      29.74576820311831, 29.8342404541355,
11      29.8774316380940] # abscisse
12 y = [-5.3906071859853, -4.2505317595919,
13      -3.10349008723970, -1.9519995345194,
14      -0.7985426911486] # ordonnées
15
16 def calcul_aire(an):
17     '''
18     renvoyer l'aire parcourue
19     '''
20     i = t_ex.index(an)
21     # calcul des aires 2020-2021
22     long1 = m.sqrt(x[i]**2 + y[i]**2)
23     long2 = m.sqrt(x[i+1]**2 + y[i+1]**2)
24     long3 = m.sqrt((x[i+1]-x[i])**2 + (y[i+1]-y[i])**2)
25     demip = (long1 + long2 + long3) / 2 # demi perimetre
26     aire = m.sqrt(demip * (demip - long1) * (demip - long2) * (demip - long3))
27     return aire
28 # A compléter
```


6.4. Etude énergétique

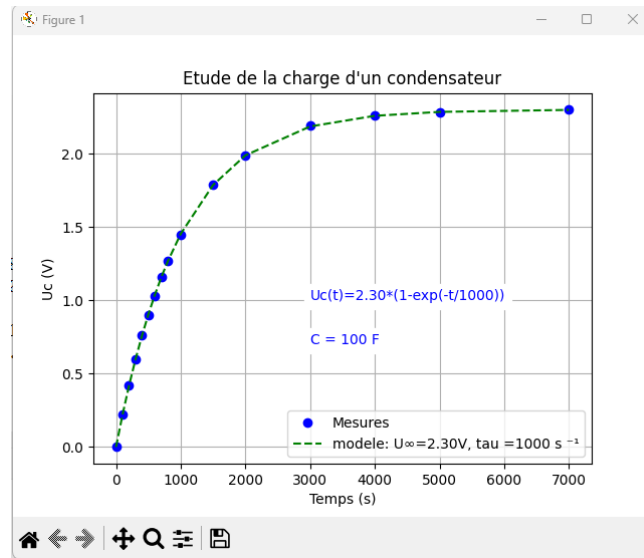
On veut représenter sur un même graphique, plusieurs séries de valeurs, calculées à partir de données expérimentales.



```
ex6-4.py x
1 # Représenter l'évolution au cours du temps des énergie cinétique, potentielle de pesanteur et mécanique
2 # d'un grélon en chute libre
3
4 import matplotlib.pyplot as plt
5
6 M = 15E-3 # masse du grélon en kg
7 G = 9.81 # Intensité de la pesanteur en N.kg-1
8 # Mesures expérimentales
9 z = [3.13, 2.92, 2.72, 2.51, 2.24, 1.90, 1.63, 1.36, 1.22] # altitude en m
10 t = [0.04, 0.08, 0.12, 0.16, 0.20, 0.24, 0.28, 0.32, 0.36] # date e s
11 v = [4.16, 4.32, 4.47, 4.76, 5.16, 5.54, 5.89, 6.11, 6.22] # vitesse en m.s-1
12 # 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 x
1  # Tracé des courbes expérimentale de la charge d'un condensateur
2  # avec un conducteur ohmique de 10Ω
3
4  import numpy as np
5  import matplotlib.pyplot as plt
6  from scipy.optimize import curve_fit
7
8  R = 10 # en ohm
9
10 # valeurs expérimentales
11 # date en seconde
12 t_ex = np.array([0, 100, 200, 300, 400, 500, 600, 700, 800,
13                  1000, 1500, 2000, 3000, 4000, 5000, 7000])
14 # tensions aux bornes du condensateur en volt
15 uc_ex = np.array([0, 0.22, 0.42, 0.6, 0.76, 0.9, 1.03, 1.16,
16                   1.27, 1.45, 1.79, 1.99, 2.19, 2.26, 2.28, 2.3])
17
18 def modele(x, a, b):
19     return a * (1 - np.exp(-x/b))
20
21 u_inf = max(uc_ex)
22 # p0 initialisation nécessaire des valeurs des parametres si le modèle n'est pas trouvé
23 (params, covariance) = curve_fit(modele, t_ex, uc_ex, p0 = [u_inf, R])
24 # A compléter
```