Python pour la physique-chimie

David THERINCOURT
Lycée Roland Garros - Académie de la Réunion
28 juillet 2022

Plus d'informations sur https://python.david-therincourt.fr

Table des matières

Chapitre 1

Introduction à Python

1.1 Qu'est-ce que Python ?

FIG. 1 – https://www.python.org/

Créer en 1991 par Guido van Rossum, Python est un langage de programmation très proche du langage algorithme (langage naturel). Cette particularité fait de Python un langage simple à apprendre et à utiliser. Performant, multiplateforme et sous licence libre, il est devenu le langage le plus utilisé au monde (devant C, C++, JAVA, ...) aussi bien dans l'éducation, la recherche et l'industrie.

L'environnement Python est très riche. En plus du langage de base, il existe une multitude de **librairies** (regroupant des modules) qui apportent à Python des fonctionnalités supplémentaires dans des domaines très variés. Par exemple, avec les trois modules **Numpy**, **Matplotlib** et **Scipy**, le langage Python est devenu une sérieuse alternative à des langages scientifiques comme Matlab ou Scilab.

Python 3.9 (2022) est la dernière version stable.

Avertissement : Il y a eu quelques changements notables au passage de Python 2 à Python 3, ce qui implique que ces deux versions ne sont pas compatibles.

1.2 Quelle distribution Python choisir ?

Une distribution Python est un **ensemble de logiciels et de librairies** qui permettent la programmation en langage Python.

Il existe une multitude de distributions Python : Anaconda, EduPython, WinPython, Portable Python, Tiny Python, ...

1.2.1 Anaconda Python

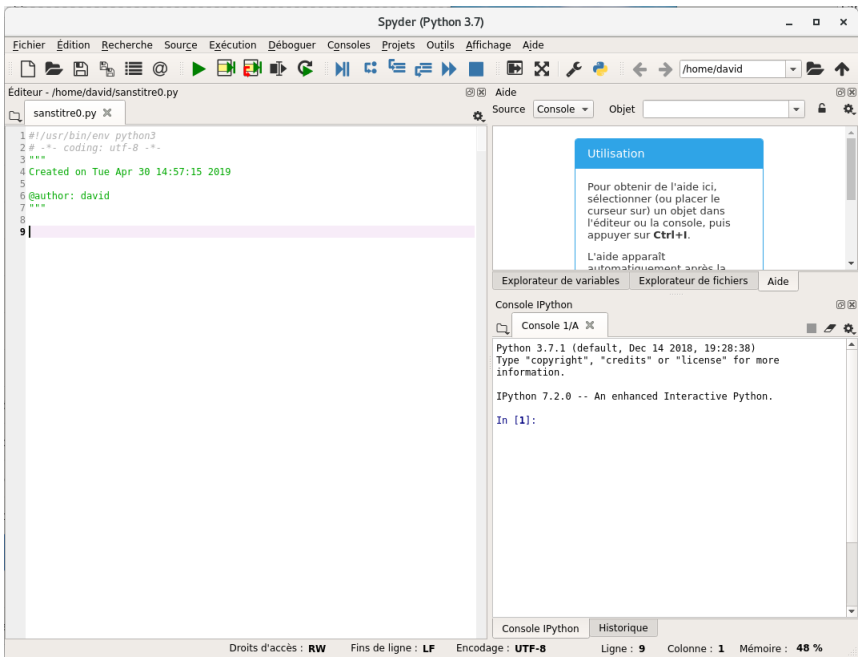
La distribution **Anaconda Python** est très populaire dans la communauté Python pour plusieurs raisons :

— multiplateforme (Windows, Linux, Mac OSX) ;
— bibliothèque étoffée ;

— outils performants (l'éditeur Spyder et bien d'autres).

FIG. 2 – https://www.anaconda.com/distribution/

Anaconda est livré avec l'environnement intégré de développement (IDE) Spyder.



L'éditeur **Spyder** est composé de plusieurs fenêtres dont :

— **la console IPython** (en bas à droite) dans laquelle les instructions Python vont être interprétées ;
— **l'éditeur de programme** (à gauche) dans lequel les instructions Python sont écrites puis enregistrées dans un fichier avec l'extension `.py`. Ce type de fichier s'appelle un **script** Python.

1.2.2 EduPython

FIG. 3 – https://edupython.tuxfamily.org/

EduPython est une distribution développée spécialement pour l'enseignement du langage Python au lycée.

Par rapport aux autres distributions classiques, EduPython présente quelques avantages non-négligeables :

— peut s'installer ou s'utiliser à partir d'une clé USB ;
— la plupart des librairies utilisées au lycée sont préinstallées ;
— s'insère plus facilement dans le réseau d'un établissement (ex. gestion du proxy pour l'accès à Internet).

EduPython propose l'éditeur PyScripteur pour l'édition de programme Python.

1.2.3 Thonny

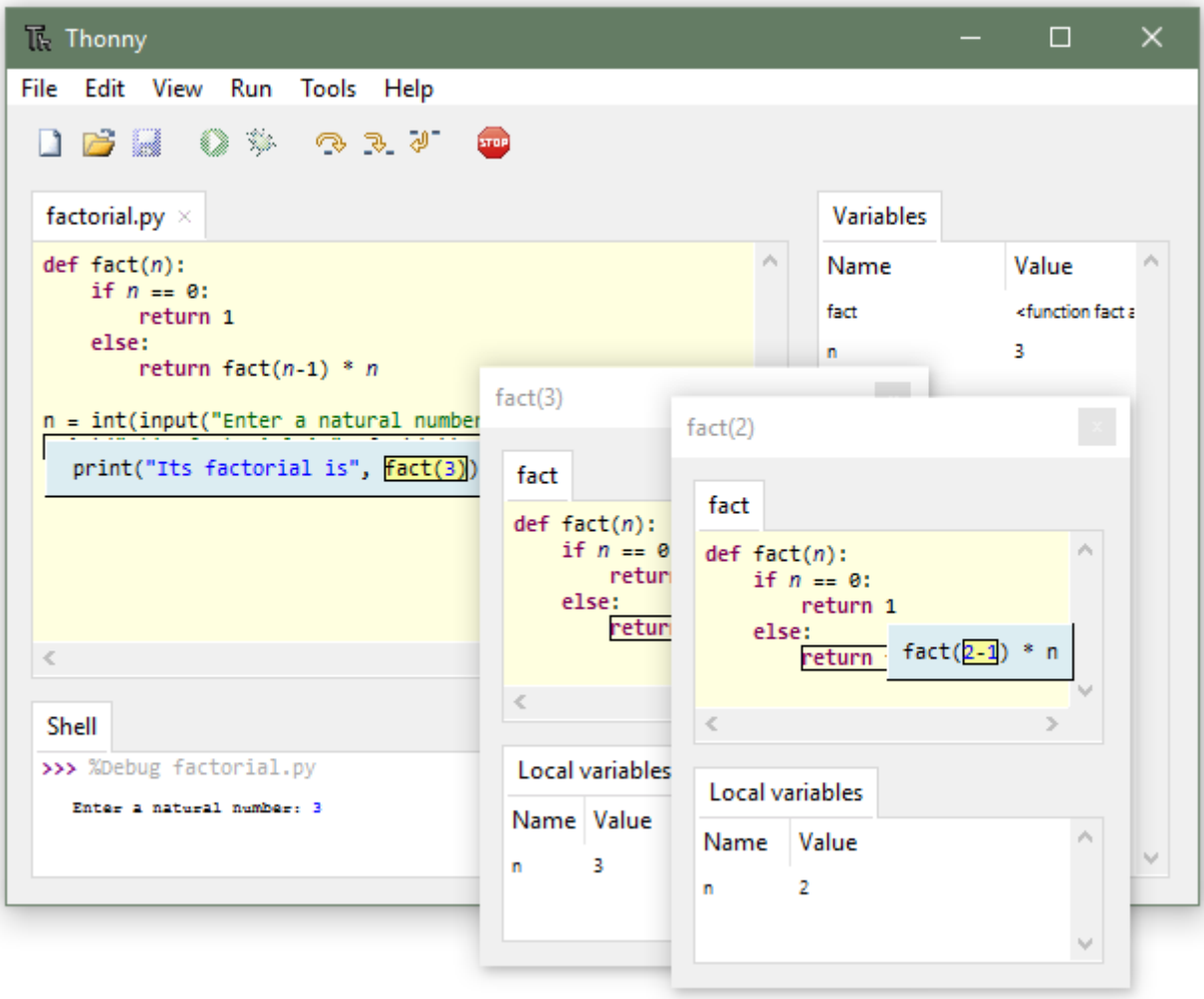


FIG. 4 – https://thonny.org/

Thonny est un éditeur **simpliste** conçu pour les débutants. Il permet surtout de changer facilement d'interpréteur Python : Anaconda Python, MicroPython pour les microcontrôleurs STM32 ou ESP32, ...

Note : La sélection de l'interpréteur Python d'Anaconda se fait à partir du menu `Exécuter > Sélectionner l'interpréteur ...`.

Ensuite sélectionner `Interpréteur Python 3 alternatif ou environnement virtuel`.

Puis chercher l'exécutable Python (`python.exe`) dans le répertoire d'installation d'Anaconda.

1.2.4 Python en ligne

Il est également possible de programmer en Python dans un navigateur Web sans aucune installation sur son ordinateur.

FIG. 5 – https://www.onlinegdb.com/online_python_compiler

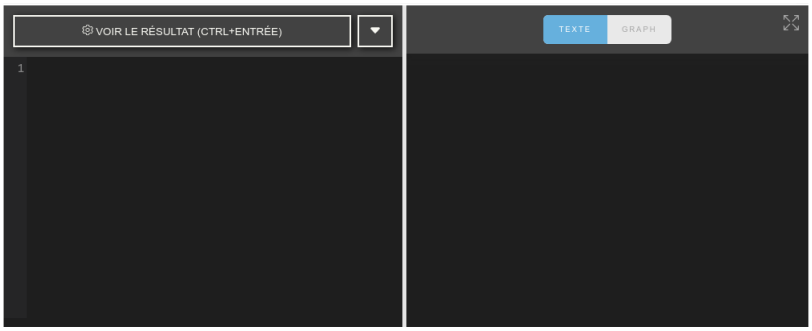


FIG. 6 – https://www.lelivrescolaire.fr/console-python

Avertissement : Attention, certaines fonctionnalités évoluées ne sont disponibles !

1.3 Premier programme Python

Voici une première instruction Python :

```
print('Bonjour')
```

Il est important de noter que cette instruction peut-être **exécutée de deux façons différentes**.

1.3.1 Directement dans la console Python

La console Python s'utilise à la manière d'une calculatrice.

FIG. 7 – Console Python de Thonny.

Les caractères >>> forme le prompt de la console Python. Cela signifie qu'une instruction Python est attendue par l'interpréteur Python.

Note : Cette technique est pratique pour faire des **tests** d'instructions ou pour **debugger** un programme.

1.3.2 A partir d'un script dans l'éditeur de texte

Les instructions Python sont sauvegardées dans un fichier texte avec l'extension `.py`. L'ensemble s'appelle un **script Python**.

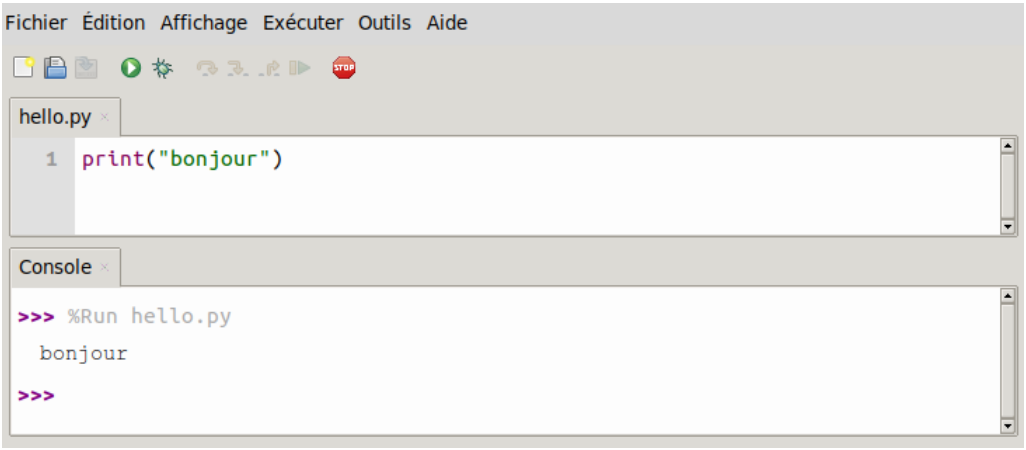


FIG. 8 – Editeur de Thonny

— Les instructions Python sont écrites séquentiellement.
— Le script est exécuté dans la console à partir du menu `Exécuter > Exécuter le script courant`, de la flèche blanche/verte ou de la touche F5.

Note : Un script sera préféré pour l'élaboration d'un programme Python comportant plusieurs lignes.

Chapitre 2

Initiation au langage Python

Cette initiation au langage Python est conçue sous la forme **d'exemples commentés**.

Avertissement : Les caractères >>> ne font pas partie des instructions Python (ne pas copier dans un programme Python). Il s'agit du **prompt** de la console Python qui signifie que l'interpréteur Python attend une (ou des) instruction(s) à exécuter !

2.1 Variables

2.1.1 Qu'est-ce qu'une variable ?

Une variable est un **emplacement mémoire** dans l'ordinateur prévu pour contenir des données.

2.1.2 Déclaration d'une variable

```
>>> a = 3
>>> nom = "Newton"
```

— Une variable est **identifiée** par un nom ;
— Le signe = est réservé à **l'affection** d'une valeur à une variable.
— Le contenu d'une variable est toujours **modifiable pendant l'exécution** du programme.

Note : Par habitude, une **constante est écrite en majuscule** même si le type constant n'existe pas en Python !

2.1.3 Affichage du contenu d'une variable

```
>>> a = 3
>>> a
3
>>> nom = "Newton"
>>> nom
'Newton'
```

— Le contenu d'une variable peut-être affiché directement dans la console Python.

```
>>> a = 3
>>> print(a)
3
>>> nom = "Newton"
>>> print(nom)
Newton
>>> print(nom, a, "Bonjour")
Newton 3 Bonjour
```

— La fonction `print()` affiche le contenu d'une ou plusieurs variables avec plus options.

2.1.4 Types d'une variable

Contrairement à d'autres langage de programmation comme C/C++ (ex. Arduino), il n'est **pas nécessaire de préciser le type d'une variable** lors de sa déclaration en langage Python. Le typage des variables est automatique (**dynamique**).

```
>>> type(10)
<class 'int'>
>>> a = 10
>>> type(a)
<class 'int'>
>>> pi = 3.14
>>> type(pi)
<class 'float'>
>>> nom = "Newton"
>>> type(nom)
<class 'str'>
```

— La fonction `type` donne le type d'une expression ou d'une variable.
— Le type `int` pour *integer* signifie que la variable contient un entier.
— Le type `float` signifie que la variable contient un flottant.
— Le type `str` pour *string* signifie que la variable contient une chaîne de caractères.

Il existe d'autres types en Python comme par exemples : `bool`, `list`, ...

2.1.5 Astuce

Il est possible de déclarer plusieurs variables sur une même ligne en utilisant la virgule comme séparateur.

```
>>> a, b = 2, 3
>>> a
2
>>> b
3
>>> a, b = b, a
>>> a
3
>>> b
2
```

Note : Cette astuce possible grâce au type `tuple` de Python !

2.1.6 Explorateur de variable

La plupart des éditeurs Python (ex. Thonny, Spyder, ...) propose un **explorateur de variables**. Toutes les valeurs des variables en cours d'interprétation sont affichées dans une fenêtre dédiée. Cette fonctionnalité est très pratique lorsque Python est utilisé comme langage scientifique.

2.2 Nombres

En Python, on distingue principalement deux types de nombres : les **entiers** et les **flottants** (nombres décimaux).

2.2.1 Les nombres entiers

```
>>> type(10)
<class 'int'>

>>> a = 10
>>> type(a)
<class 'int'>
```

— La fonction `type()` donne le type d'une expression ou d'une variable.
— Le mot clé `int` pour *integer* signifie que la variable contient un entier.

Les **opérateurs mathématiques** classiques s'appliquent.

Opérateurs mathématiques	
`+`	Addition
`-`	Soustraction
`*`	Multiplication
`/`	Division
`//`	Quotient de la division entière
`%`	Reste de la division entière
`**`	Élever à la puissance

```
>>> b = 3
>>> a+b
13
>>> a/b
3.3333333333333335
>>> a//b
3
>>> a%b
1
```

— L'opérateur `//` donne le **quotient** de la division entière.
— L'opérateur `%` (modulo) donne le **reste** de la division entière.

```
>>> 2**100
1267650600228229401496703205376
```

— En Python, la **taille d'un entier n'est pas limitée** !

2.2.2 Les nombres flottants

Un flottant est un **nombre décimal** (nombre à virgule).

```
>>> type(9.80665)
<class 'float'>
```

— Le type `float` pour les nombres à virgule flottante.

```
>>> g = 9.80665
>>> round(g,2)
9.81
>>> m = 25
>>> P = m*g
>>> print(P)
245.16625
```

— La fonction `round(x, n)` arrondie la valeur flottante `x` à `n` chiffres après la virgule.

Note : La type `float` définit les notations spéciales `inf` (**infini**) et `nan` (**not a number**). Cette dernière est très pratique pour des valeurs non définies dans des tableaux de données.

```
>>> float("nan")
```

2.2.3 Les nombres complexes

Le langage Python gère également les nombres complexes.

```
>>> z = 5+6j
>>> type(z)
<class 'complex'>
>>> abs(z)
7.810249675906654
>>> 1j**2
(-1+0j)
```

— Le nombre imaginaire est noté `1j` !
— La fonction `abs()` donne le module.

Exercices d'application

Exercice 1. (dilution d'une eau salée)

2.3 Textes

En programmation, un texte est représenté par une **chaîne de caractères** (suite de caractères).

2.3.1 Syntaxe

En langage Python, les chaînes de caractères sont toujours **délimitées par les caractères `'` ou `"`**.

```
>>> type("Bonjour")
<class 'str'>
>>> ch1 = "Bonjour"
>>> print(ch1)
Bonjour
```

— Le type `str` pour *string* (chaîne de caractères).

2.3.2 Concaténation

```
>>> ch2 = "Paul"
>>> ch1 + ch2
'BonjourPaul'
>>> ch3 = ch1 + " " +  ch2
>>> print(ch3)
Bonjour Paul
```

— L'opérateur + réalise la **concaténation** de chaînes de caractères.

```
>>> m = 50
>>> g = 9.81
>>> P = m*g
>>> reponse = 'Une masse de ' + m + ' kg a un poids de ' + P + ' N sur Terre !'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> reponse = 'Une masse de ' + str(m) + ' kg a un poids de ' + str(P) + ' N sur Terre !
 ↪'
>>> reponse
'Une masse de 50 kg a un poids de 490.5 N sur Terre !'
```

— Il n'est pas possible de concaténer des chaînes de caractères avec d'autres types !
— La fonction `str()` permet la **conversion** de n'importe quel type en chaîne de caractères.

2.3.3 Afficher du texte avec la fonction `print()`

Par contre, la fonction `print()` affiche n'importe quel type en texte.

```
>>> m = 50
>>> g = 9.81
>>> P = m*g
>>> print('Une masse de ', m, ' kg a un poids de ', P, ' N sur Terre !')
Une masse de  50  kg a un poids de  490.5  N sur Terre !
```

— Les différents types sont séparés par une virgule `,`.
— A l'affichage, un espace est ajouté à chaque virgule.

2.3.4 Saisir un texte avec la fonction `input()`

Avertissement : Lorsque Python est utilisé comme un langage de programmation scientifique, il n'est pas nécessaire de saisir le contenu d'une variable au clavier. Il est **plus pratique de changer la valeur d'une variable directement dans le code**. L'utilisation de la fonction `input()` n'est pas conseillée en physique-chimie sauf cas particulier.

En python, la fonction `input()` demande à l'utilisateur du programme de saisir un texte au clavier.

```
>>> rep = input()
Bonjour
>>> rep
'Bonjour'
```

— La fonction `input()` **renvoie la chaîne de caractères saisie au clavier** par l'utilisateur.
— La touche **Entrer** valide la fin de la saisie du texte au clavier.
— Ici la chaîne de caractères est affectée à la variable `rep`.

```
>>> nom = input('Quel est votre nom ? ')
Quel est votre nom ? David
>>> nom
'David'
```

— Il est possible d'ajouter un texte pour aider l'utilisateur lors de la saisie.

```
>>> n = input('Entrer un entier : ')
Entrer un entier : 5
>>> n
'5'
>>> n*3
'555'
```

— Attention, la fonction `input()` en **renvoie toujours une chaîne de caractères** !

```
>>> rep = input('Entrer un entier : ')
Entrer un entier : 5
>>> rep
'5'
>>> n = int(rep)
>>> n
5
>>> n*3
15
```

— La fonction `int()` convertit une chaîne de caractères décrivant un entier en un type entier.

```
>>> n = int(input('Entrer un entier : '))
Entrer un entier : 5
>>> n
5
>>> n*3
15
```

— Il est possible de combiner les fonctions `int()` et `input()` sur la même ligne.

Note : De la même manière, la fonction `float()` permet la conversion en type flottant.

Exercices d'application

Exercice 1 (dilution)

2.4 Booléens

Un booléen est un type de variable logique à deux états : **Vrai** ou **Faux**.

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

— En langage Python, un booléen prend les valeurs `True` (vrai) ou `False` (faux).

2.4.1 Opérateurs de comparaison

Opérateurs de comparaison	
>	Strictement supérieur
<	Strictement inférieur
<=	Inférieur ou égal
>=	Supérieur ou égal
==	Égal à
!=	Différent de

```
>>> 3>2
True
>>> 3<=2
False
>>> 3 == 2
False
```

— Les opérateurs de comparaison renvoient toujours un booléen (`True` ou `False`)

2.4.2 Opérateurs logiques

Opérateurs logiques	
`and`	ET logique
`or`	OU logique
`not`	NON logique

```
>>> True and True
True
>>> True and False
False
>>> not False
True
```

— Les mots réservés `and` et `not` sont des opérateurs logiques.

2.5 Conditionnelles

Les structures conditionnelles permettent de faire des tests.

2.5.1 Condition `if`

```
if <condition>:
   <bloc instruction(s) pour condition vraie>
```

L'instruction `if` évalue une expression conditionnelle puis exécute le bloc d'instruction qui suit uniquement si la condition est vraie.

Avertissement : En langage Python, un bloc d'instruction doit être toujours **indenté** (décalage de 4 caractères par convention) par rapport au reste du code.

```
a = 3
b = 5

if a>b:
   print("AAA")

if a>b:
   print("BBB")

if a>b:
print("CCC")
```

```
>>> %Run
BBB
CCC
```

— Ici `a>b` et `a<b` sont les expressions conditionnelles à évaluer.
— Le caractère `:` marque le début du bloc d'instruction à exécuter.
— La fonction `print("AAA")` n'est pas exécutée car l'expression `a>b` renvoie `False`.
— La fonction `print("BBB")` est exécutée car l'expression `a<b` renvoie `True`.
— La fonction `print("CCC")` est exécutée car elle n'est pas **indenté** donc pas dans le bloc du `if` !

2.5.2 Condition `if else`

```
if <condition>:
   <bloc instruction(s) pour condition vraie>
else:
   <bloc instruction(s) pour condition fausse>
```

L'instruction `if` évalue toujours l'expression conditionnelle puis exécute un bloc d'instruction si la condition est vraie. Puis l'instruction `else` exécute un autre bloc d'instructions pour cette condition fausse.

```
a = 3
b = 5

if a>b:
   print("a strictement plus grand que b !")
else:
   print("a plus petit ou égal à b !")
```

```
>>> %Run
a plus petit ou égal à b !
```

2.5.3 Condition `if elif else`

Permet de tester plusieurs conditions suivant la structure ci-dessous.

```
if <condition_1>:
   <bloc instruction(s) pour condition_1 vraie>
elif <condition_2>:
   <bloc instruction(s) pour condition_1 fausse et condition 2 vraie>
else:
   <bloc instruction(s) pour toutes les conditions précédentes fausses>
```

```
a = 3
b = 5

if a>b:
   print("a strictement plus grand que b !")
elif a<b:
   print("a strictement plus petit que b !")
else:
   print("a est égal à b !")
```

```
>>> %Run
a strictement plus petit que b !
```

Exemples au programme de physique-chimie

Chute complète (élastique tendu) (première générale)

Dosage par titrage par suivi conductimétrie (terminale générale)

2.6 Tableaux (listes)

En sciences physiques, les tableaux de données sont souvent présents.

En langage Python, les tableaux sont représentés par les **listes** (type `list`).

2.6.1 Construire une liste

```
>>> l = [0, 'a', 4.13]
>>> l
[0, 'a', 4.13]
```

— Une liste est **délimitée** par des crochés `[]` ;
— Les éléments d'une liste sont **séparés** par des virgules ;
— Une liste peut contenir des **types différents** (mais généralement des nombres en sciences physiques).

2.6.2 Position d'un élément dans une liste

```
>>> l[0]
0
>>> l[1]
'a'
```

(suite sur la page suivante)

(suite de la page précédente)

```
>>> l[2]
4.13
>>> l[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

— Chaque élément d'une liste est repéré par un **indice** (position dans la liste).
— L'indice du **premier élément** est toujours `0` !

```
>>> l[-1]
4.13
>>> l[-2]
'a'
>>> l[-4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

— Les **indices négatifs** offrent la possibilité de parcourir la liste depuis la fin.
— L'indice `-1` étant le **dernier élément** de la liste !

2.6.3 Modification d'un élément

```
>>> l[1] = 3
>>> l
[0, 3, 4.13]
```

— Un élément d'une liste peut-être **modifié** par une affectation.
— L'élément à modifier est toujours repéré par son indice.

2.6.4 Sélection d'éléments

```
>>> x = [0, 1, 2, 3, 4, 5, 6]
>>> x[2:]
[2, 3, 4, 5, 6]
>>> x[:3]
[0, 1, 2]
>>> x[2:5]
[2, 3, 4]
```

— Le caractère `:` permet de **sélectionner** d'une partie de la liste.

2.6.5 Taille d'une liste avec `len`

```
>>> y = [0, 1, 2, 6, 5, 4, 3]
>>> len(y)
7
```

— La fonction standard `len()` retourne le nombre d'éléments d'une liste.

2.6.6 Ajouter un élément avec `append`

```
>>> y = [0, 1, 2, 6, 5, 4, 3]
>>> y.append(9)
>>> y
[0, 1, 2, 6, 5, 4, 3, 9]
```

— La méthode `append()` ajoute un élément à la fin de la liste.
— Application de cette méthode modifie la liste sur laquelle elle est appliquée !

2.6.7 Autres opérations

```
>>> y = [0, 1, 2, 6, 5, 4, 3]
>>> dir(y)
[... , 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
 ↪'reverse', 'sort']

>>> help(y.pop)
Help on built-in function pop:

pop(index=-1, /) method of builtins.list instance
   Remove and return item at index (default last).

   Raises IndexError if list is empty or index is out of range.
```

— La fonction `dir(y)` liste l'ensemble des méthodes applicables sur la liste `y`.
— La fonction `help(y.pop)` affiche l'aide de la méthode `pop()` appliquée à la liste `y`.

Exercices d'application

Exercice 2. (distance focale d'une lentille)

2.7 Boucle for

En programmation, une boucle est une **structure répétant plusieurs fois** (ou indéfiniment) un bloc d'instructions.

La boucle `for` s'utilise lors que le **nombre d'itérations est connu à l'avance**. Il s'agit d'une **boucle bornée**.

2.7.1 Parcours d'une liste

```
for <variable> in <liste>:
    <bloc instructions>
```

Le parcours séquentiel d'une liste se fait suivant la structure précédente.

```
for i in [1, 2, 3]:
   print(i)
```

```
>>> %Run
1
2
3
```

— La variable `i` prend itérativement les valeurs 1, 2 et 3 dans la liste `[1, 2, 3]`.
— Le nombre d'itérations d'une boucle de type `for` est toujours connu à l'avance !

2.7.2 Fonction `range()`

La fonction `range()` est pratique pour générer automatiquement une liste de parcours.

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

— La fonction `range(n)` renvoie un itérateur sur une suite de valeurs entières entre `0` à `n` (non incluse).

Avertissement : Dans cet exemple, la fonction `list()` convertit l'itérateur renvoyé par la fonction `range()` en une liste pour une meilleure compréhension de son fonctionnement. Par la suite, la fonction `list()` ne sera pas utile !

2.7.3 Structure `for i in range(n)`

C'est la structure de la boucle `for` la plus souvent rencontrée en langage Python.

```
for i in range(5):
    print(i)
```

```
>>> %Run
0
1
2
3
4
```

— La fonction `range(n)` facilite la création d'une boucle `for` en renvoyant un itérateur de `0` à `n` (non inclus).
— La variable `i` (entier) est un **compteur** et prend la valeur suivante de l'itérateur à chaque itération de la boucle.
— La boucle `for` effectue une **dernière itération** lorsque `i` prend la dernière valeur de l'itérateur c.a.d. `n-1`. Puis la boucle s'arrête et le programme peut continuer.
— Au final, cette boucle réalise bien `n` itérations.

2.7.4 Autres variantes de la fonction `range()`

2.7.4.1 Avec la forme `range(a, b)`

```
for i in range(2, 5):
    print(i)
```

```
>>> %Run
2
3
4
```

— `a` est le premier entier.
— `b` est le dernier entier non inclus.

2.7.4.2 Avec la forme `range(a, b, p)`

```
for i in range(2,9,3):
   print(i)
```

```
>>> %Run
2
5
8
```

— `a` est le premier entier.
— `b` est le dernier entier non inclus.
— `p` est le pas d'incrémentation.

2.7.5 Construction d'une liste en compréhension

En Python, il est possible de construire une **liste en compréhension**, c.a.d. à partir d'une boucle `for` sur une seule ligne avec la syntaxe ci-dessous.

```
[<expression_element> for i in range(n)]
```

— `<expression_element>` est une expression qui définit le calcul de l'élément en cours de la liste à chaque itération de la boucle `for`.
— Le compteur `i` est utilisable dans cette expression.

```
>>> l = [2*i+1 for i in range(5)]
>>> l
[1, 3, 5, 7, 9]
```

Exercices d'application

Exercice 3 (célérité du son)

Exercice 4 (vitesse d'une balle de tennis)

Exercice 5 (énergies d'une balle de tennis)

Exercice 6 (énergies d'une balle de tennis par compréhension de liste)

Exemples au programme de physique-chimie

Mouvement d'un point : vecteur vitesse (classe de seconde)

Mouvement d'un point : vecteur variation vitesse (classe de première)

Titrage - Evolution des quantités de matière (classe de terminale)

2.8 Boucle while

2.8.1 Principe

```
while <condition>:
   <bloc instructions>
```

La boucle `while` est utilisée lorsque le **nombre d'itérations n'est pas connu à l'avance**.

2.8.2 Exemple : saisie au clavier

Le programme qui ci-dessous demande à un utilisateur de saisir au clavier la réponse à la question suivante :

Quelle est la couleur du cheval blanc d'Henry IV ?

Il n'est pas possible de savoir à l'avance si l'utilisateur du programme va donner la bonne réponse dès la première saisie au clavier. Il faudra donc lui demander de saisir sa réponse tant que cette dernière est différente de `"blanc"`.

```
reponse = ''
while reponse != 'blanc':
   reponse = input("Quelle est la couleur du cheval blanc d'Henry IV ? ")
print("Bonne réponse !")
```

```
>>> %Run exemple.py
Quelle est la couleur du cheval blanc d'Henry IV ? Blanc
Quelle est la couleur du cheval blanc d'Henry IV ? BLANC
Quelle est la couleur du cheval blanc d'Henry IV ? blanc
Bonne réponse !
```

— Il est important d'**initialiser la variable** `reponse` avec une mauvaise valeur avant le `while`. Sinon il n'est pas possible d'entrer dans la boucle !
— La boucle s'effectue bien indéfiniment tant que la réponse n'est pas la bonne donc tant que la condition `reponse != "blanc"` (réponse différent de blanc) est vraie !

Exercices d'application

Exercice 7 (nombre de dilutions)

Exercice 8 (avancement d'une réaction chimique)

Exemples au programme de physique-chimie

Évolution d'un système chimique (classe de première)

Titrage - Evolution des quantités de matière (classe de terminale)

2.9 Fonctions

2.9.1 Principe

En programmation, une fonction **réalise un traitement** à partir de variables d'entrées (arguments) puis **renvoie le résultat** de ce traitement.

Avertissement : Une fonction peut admettre **aucun, un ou plusieurs arguments.**

2.9.2 Fonction à un argument

La fonction ci-dessous calcule la surface de l'aire d'un carré de côté `a`.

```
def aire_carre(a) :
   aire = a*a
   return aire
```

— Le mot clé `def` est toujours utilisé pour définir une fonction.
— Ici `aire_carre` est le **nom (identifiant) de la fonction**.
— La variable `a` est un **paramètre** (ou argument) de la fonction.
— Après le caractère `:` toutes instructions appartenant à la fonction doivent-être **indentées**.
— Le mot clé `return` est utilisé pour **renvoyer le résultat** de la fonction.

```
>>> %Run
>>> aire_carre(5)
25
```

— L'exécution du script est indispensable avant l'**appel** de la fonction.
— Les **parenthèse sont obligatoires** afin de préciser l'argument lors de l'appel de la fonction.

2.9.3 Fonction à deux arguments

Considérons maintenant une fonction qui retourne la surface de l'aire d'un rectangle de longueur `L` et de largeur `l`.

```
def aire_rectangle(L, l):
    aire = L*l
    return aire
```

— Ici la fonction prend 2 arguments `l` et `L` !

```
>>> %Run
>>> aire_rectangle(5, 4)
20
```

La variable intermédiaire `aire` n'est indispensable. L'écriture de la fonction peut se simplifier comme suit :

```
def aire_rectangle(L, l):
    return L*l
```

2.9.4 Exemple : énergie mécanique

La fonction ci-dessous calcule une énergie potentielle de pesanteur.

```
def Epp(m, h):
    return m*9.81*h
```

— Les variables `m` et `h` sont locales. Elles n'existent que dans la fonction !

```
>>> %Run
>>> Epp(50,10)
4905.0
```

Il est possible de calculer l'énergie mécanique à partir des fonctions donnant l'énergie potentielle et l'énergie cinétique.

```
def Epp(m, h):
    return m*9.81*h

def Ec(m, v):
    return 0.5*m*v**2

def Em(m, h, v):
    return Epp(m, h) + Ec(m, v)
```

— La fonction `Em()` fait bien appel aux deux autres fonctions `Epp()` et `Ec()` !

```
>>> %Run
>>> Epp(50, 10)
4905.0
>>> Ec(50, 13)
4225.0
>>> Em(50, 10, 13)
9130.0
>>> m
Traceback (most recent call last):
...
NameError: name 'm' is not defined
```

— Pour chaque fonction, la variable `m` n'est pas la même. C'est toujours une **variable locale** de la fonction considérée. Elle n'est d'ailleurs pas accessible dans le programme principal !

Exercices d'application

Exercice 9 (dilution)

2.10 Modules, paquets et librairies

Le langage Python est fourni de base avec une palette d'outils (fonctions, méthodes, ...) qui est insuffisante dans certaines situations.

```
>>> sqrt(2)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
```

Dans cet exemple, il n'est pas possible d'effectuer une racine carrée avec Python de base !

Pour y remédier, des **outils supplémentaires sont implémentés** dans des modules de la **librairie standard** (ex. `math`, `csv`, `statistics`, ...) ou des modules de **librairies tierces** (ex. `numpy`, `matplotlib`, `scipy`, ...) développées par la communauté Python.

2.10.1 Importation d'un module

Par exemple, le module `math` de la librairie standard intègre la plupart des fonctions mathématiques.

```
>>> import math
>>> sqrt(2)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
```

— L'instruction `import` charge le module `math`.
— La fonction `sqrt()` n'est toujours pas disponible bien que le module `math` soit chargé !

Il faut donc préciser le nom du module lors de l'appel d'une composante (fonction, ...) de ce module.

```
>>> import math
>>> math.sqrt(2)
1.4142135623730951
```

— La syntaxe `math.sqrt()` signifie que `sqrt()` est une fonction du module `math`.

— Toutes les autres fonctions mathématiques du module `math` sont ainsi disponibles de cette manière.

Il est parfois pratique de renommer le module lorsque son nom est trop long à écrire à chaque fois !

```
>>> import math as mt
>>> mt.sqrt(2)
1.4142135623730951
>>> mt.exp(1)
2.718281828459045
```

— Dans cet exemple, `mt` est un **alias** de `math`.

2.10.2 Autres possibilités d'importation

Il est envisageable d'importer une seule fonction (ou plusieurs) d'un module.

```
>>> from math import sqrt
>>> sqrt(2)
1.4142135623730951
```

— Seule la fonction `sqrt()` a été importée.
— Son appel se fait directement sans préciser le nom du module.

Ou d'importer tous les composantes d'un modules à l'aide du caractère `*`.

```
>>> from math import *
>>> sqrt(3)
1.7320508075688772
>>> exp(1)
2.718281828459045
```

Avertissement : Cependant cette pratique est **à éviter** tant que possible !

2.10.3 Aide sur un module

La fonction `dir()` liste toutes les outils disponibles pour un module.

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',
 ↪ 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees
 ↪', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod',
 ↪'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan
 ↪', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan',
 ↪'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt
 ↪', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

— Les fonctions (ou méthodes) dont l'identifiant est encadrées par les caractères `___` sont des fonctions cachées.

Et la fonction `help()` renvoie une aide détaillée du module ou d'une fonction du module.

```
>>> help(math)
Help on built-in module math:

NAME
   math
```

(suite sur la page suivante)

(suite de la page précédente)

```
DESCRIPTION
   This module provides access to the mathematical functions
   defined by the C standard.

FUNCTIONS
   acos(x, /)
      Return the arc cosine (measured in radians) of x.

      The result is between 0 and pi.

   acosh(x, /)
      Return the inverse hyperbolic cosine of x.

...
```

```
>>> help(math.sqrt)
Help on built-in function sqrt in module math:

sqrt(x, /)
   Return the square root of x.
```

2.10.4 Modules pour les sciences physiques

Plusieurs **modules**, **paquets** (ensemble de modules) et **librairies** (ensemble de paquets) sont souvent rencontrés en sciences physiques tels que `math`, `numpy`, `matplotlib`, `scipy`, ...

Code source 1 – Exemples d'importation.

```
>>> from math import log10
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from scipy.stats import linregress
```

Exercices d'application

Exercice 10 (fonction pH)

Chapitre 3

Calcul scientifique avec Python

3.1 Modules et librairies pour les sciences

En Python, des librairies sont spécifiquement développées pour le calcul scientifique.

Pour commencer, cette section résume les fonctionnalités des différentes librairies les plus rencontrées. Ces dernières sont présentées plus en détail dans la suite de cette documentation.

3.1.1 Math

Le module `math` fait partie de la **librairie standard** de Python.

Fonction/constante	Description
`math.sqrt`	Fonction racine carrée
`math.sin`	Fonction sinus (angle en radian)
`math.cos`	Fonction cosinus (angle en radian)
`math.exp`	Fonction exponentielle
`math.log10`	Fonction logarithme décimal
`math.pi`	Constante Pi
`math.nan`	Valeur flottante non définie (Not A Number)

3.1.2 Numpy

Le paquet (ensemble de modules) `Numpy` facilite la manipulation de tableaux avec Python.

```
import numpy as np
```

L'importation de ce paquet se fait usuellement avec alias `np`.

TABLEAU 1 – Plusieurs méthodes pour créer des tableaux Numpy.

Fonction/méthode	Description
np.array	Tableau Numpy à partir d'une liste Python.
np.linspace	Tableau Numpy à partir d'un intervalle de N valeurs.
np.arange	Tableau Numpy à partir d'un intervalle avec un pas d'incrémentation.
np.zeros	Tableau Numpy de zéros de taille N.
np.loadtxt	Tableau(x) Numpy à partir de l'importation d'un fichier CSV

TABLEAU 2 – Quelques exemples de fonctions et constantes mathématiques.

Fonction/constante	Description
np.sin	Fonction sinus
`np.exp`	Fonction exponentielle
`np.sqrt`	Fonction racine carré
`np.pi`	Constante pi
`np.nan`	Valeur flottante non définie (Not A Number)

TABLEAU 3 – Statistique

Fonction/méthode	Description
np.mean	Valeur moyenne d'un tableau Numpy
np.std	Ecart-type d'un tableau Numpy
np.random.normal	Génération de nombres aléatoires suivant la loi normale

TABLEAU 4 – Autres traitements.

Fonction/méthode	Description
np.polyfit	Modélisation à partir d'un polynôme d'ordre n
`np.trapz`	Intégration d'un tableau avec la méthode des trapèzes.
`np.fft`	Transformé rapide de Fourier (spectre)

3.1.3 Matplotlib

`Matplotlib` est une librairie (ensemble de paquets) très complète pour les représentations graphiques. En particulier, le module `matplolib.pyplot` simplifie son usage.

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

L'importation du module `pyplot` se fait habituellement avec alias `plt`.

TABLEAU 5 – Les bases pour les graphiques.

Fonction/méthode	Description
plt.plot	Trace une courbe à partir d'une série de points
plt.show	Affiche la figure (à appeler en dernier !)

TABLEAU 6 – Paramétrage du repère.

Fonction/méthode	Description
plt.title	Ajoute un titre au repère.
plt.xlabel	Ajoute une légende à l'axe des abscisses.
plt.ylabel	Ajoute une légende à l'axe des ordonnées.
plt.xlim	Fixe l'échelle sur l'axe des abscisses.
plt.ylim	Fixe l'échelle sur l'axe des ordonnées.
plt.grid	Ajoute un quadrillage.

TABLEAU 7 – Autres types de représentations graphiques.

Fonction/méthode	Description
plt.scatter	Trace une nuage de points (peu utilisée au profit de `plt.plot`)
plt.hist	Trace un histogramme.
plt.quiver	Trace un champ de vecteurs (ou un vecteur).
`plt.stem`	Trace des tiges (représentation d'un spectre).

TABLEAU 8 – Annotations.

Fonction/méthode	Description
`plt.text`	Ajoute un texte simple aux coordonnées `(x, y)` d'un repère
`plt.arrow`	Dessine une flèche dans un repère.
`plt.annotate`	Ajoute un texte aux coordonnées `(x, y)` avec plus d'options (ex. flèche)

TABLEAU 9 – Droites et zones de délimitation

Fonction/méthode	Description
`plt.axvline`	Trace une droite verticale à l'abscisse `x0`
`plt.axhline`	Trace une droite horizontale à l'ordonnée `y0`.
`plt.axvspan`	Délimite une zone rectangulaire verticale entre `x1` et `x2`.
`plt.axhspan`	Délimite une zone rectangulaire horizontale entre `y1` et `y2`.
`plt.fill_between`	Remplit l'aire entre deux courbes (ou entre une courbe et l'axe des abscisses).

3.1.4 Scipy

La librairie `Scipy` comporte un grand nombre de fonctions évoluées pour le traitement numérique regroupées dans des modules spécifiques (`stats`, `interpolate`, ...)

TABLEAU 10 – Modélisation

Fonction/méthode	Description
scipy.stats.linregress	Modélisation par régression linéaire
scipy.optimize.curve_fit	Modélisation par une fonction quelconque
scipy.interpolate.interp1d	Interpolation polynomiale

Code source 1 – Exemples d'importation.

```
from scipy.stats import linregress
from scipy.optimize import curve_fit
from scipy.interpolate import interp1d
```

TABLEAU 11 – Pour le supérieur (CPGE, ...)

Fonction/méthode	Description
scipy.integrate.odeint	Intégration (résolution) d'une fonction décrivant une équation différentielle.
`scipy.optimize.bisect`	Détermination des racines d'une fonction par la méthode dichotomique

Code source 2 – Importation.

```
from scipy.integrate import odeint
from scipy.optimize import bisect
```

3.1.5 Statistics

Statistics est un module de la librairie standard de Python pour des calculs statistiques sur des tableaux de données numériques.

TABLEAU 12 – Principales fonctions

Fonction	Description
statistics.mean	Valeur moyenne arithmétique.
statistics.stdev	Ecart type.

Code source 3 – Exemple d'importation.

```
from statistics import mean, stdev
```

3.1.6 Random

Le module **random** fait également partie de la librairie standard de Python. Il implémente des fonctions de génération de nombres aléatoires.

TABLEAU 13 – Principales fonctions

Fonction	Description
random.randint	Retourne un entier au hasard dans un intervalle.

Code source 4 – Exemples d'importation

```
from random import randint
```

3.2 Tableaux Numpy

3.2.1 Chargement de Numpy

```
>>> import numpy as np
```

— Le paquet `numpy` est habituellement importé avec l'alias `np` qui est plus rapide à écrire à chaque fois !

3.2.2 Créer des tableaux Numpy

3.2.2.1 A partir d'une liste avec `array()`

```
>>> a = np.array([1, 2, 3, 4])
>>> a
array([1, 2, 3, 4])
>>> print(a)
[1 2 3 4]
```

— Il est possible de créer un tableau (ici à 1 dimension) à partir d'une liste.

3.2.2.2 A partir d'un intervalle et d'un nombre d'éléments avec `linspace()`

```
>>> a = np.linspace(1, 7, 3)
>>> a
array([1., 4., 7.])
```

— La fonction `linspace(start,end, nb)` génère `n` valeurs entre `start` et `end`.

3.2.2.3 A partir d'un intervalle et d'un pas avec `arange()`

```
>>> a = np.arange(1, 2, 0.2)
>>> print(a)
[1.  1.2 1.4 1.6 1.8]
```

— La fonction `arange(a,b,p)` construit un tableau Numpy de `a` à `b` (non compris) avec un pas de `p`.

3.2.2.4 A partir de zéros avec `zeros()`

Il est parfois intéressant de créer un tableau de zéros dont les valeurs pourront être modifiées par la suite.

```
>>> a = np.zeros(5)
>>> print(a)
[0. 0. 0. 0. 0.]
```

3.2.3 Manipulation des tableaux Numpy

```
>>> a = np.array([1, 2, 3, 4])
>>> a*3
array([ 3,  6,  9, 12])
>>> a**2
array([ 1,  4,  9, 16])
```

— Les opérations mathématiques s'appliquent **itérativement** sur les tableaux de type `Numpy`.

```
>>> l = [1, 2, 3, 4]
>>> l*3
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

— Ce n'est pas le cas avec les listes !

```
>>> a = np.array( )
>>> b = np.array([5, 6, 3, 8])
>>> 3*a + b
array([ 8, 12, 12, 20])
>>> a == b
array([False, False,  True, False])
```

— La plupart des opérateurs sont disponibles avec les tableaux Numpy !

3.2.4 Fonctions et tableaux Numpy

```
>>> a = np.array( )
>>> import math
>>> math.sqrt(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: only size-1 arrays can be converted to Python scalars
```

— Par contre, il n'est pas possible d'appliquer les fonctions mathématiques du module `math`.

```
>>> np.sqrt(a)
array([1.        , 1.41421356, 1.73205081, 2.        ])
>>> np.exp(a)
array([ 2.71828183,  7.3890561 , 20.08553692, 54.59815003])
```

— Mais heureusement le paquet `Numpy` intégre ses propres fonctions mathématiques.

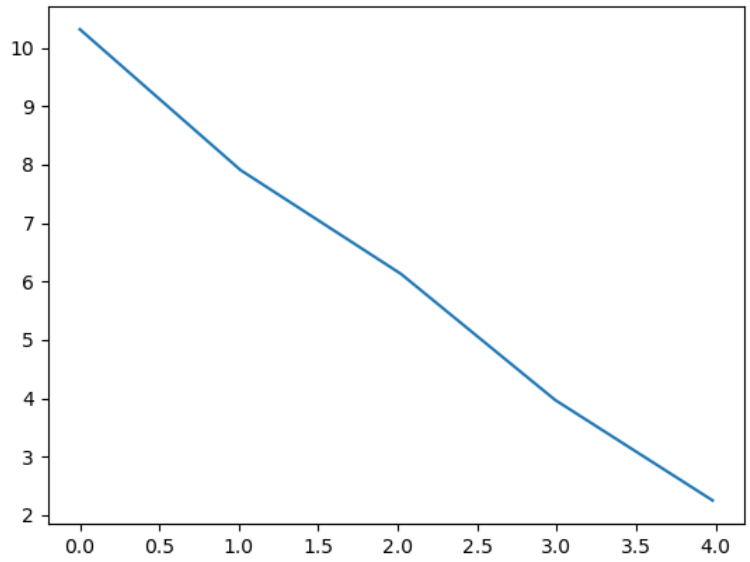
3.3 Tracer un nuage de points

3.3.1 Fonction `plot()`

La fonction `plot()` du module `matplotlib.pyplot` est plus souvent utilisée pour représenter un nuage de points.

3.3.1.1 Les bases

```
import matplotlib.pyplot as plt

x = [0, 1.01, 2.02, 2.99, 3.98]     # Données en abscisse
y = [10.32, 7.91, 6.13, 3.97, 2.25] # Données en ordonnée

plt.plot(x, y)       # Tracé de la courbe
plt.show()          # Affichage de la courbe
```

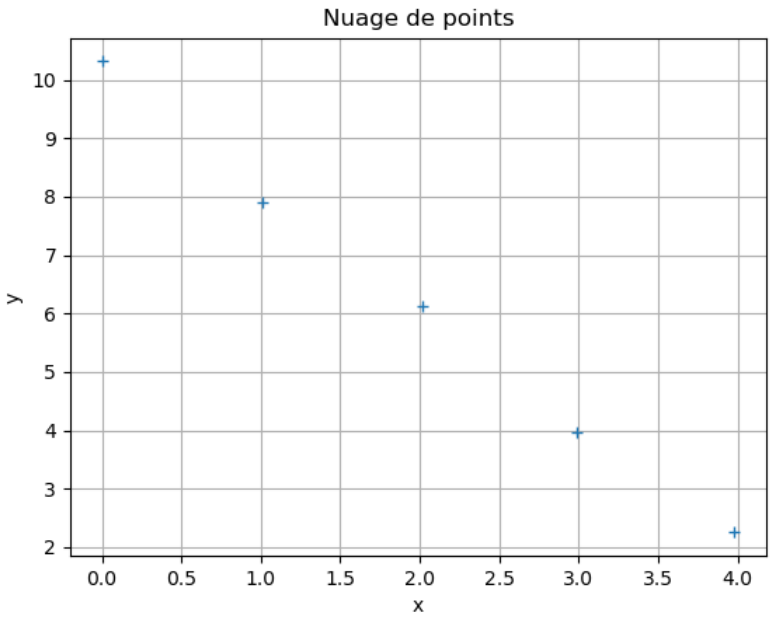


— Le module `pyplot` de la librairie `matplotlib` est importée avec l'alias `plt`.
— La fonction `plot()` trace un nuage de points à partir des listes `x` et `y`.
— La fonction `show()` appelée à la fin affiche la fenêtre graphique.

Avertissement : Par défaut la fonction `plot()` trace un segment entre deux points consécutifs.

3.3.1.2 Ajouter un titre, une légende et une grille

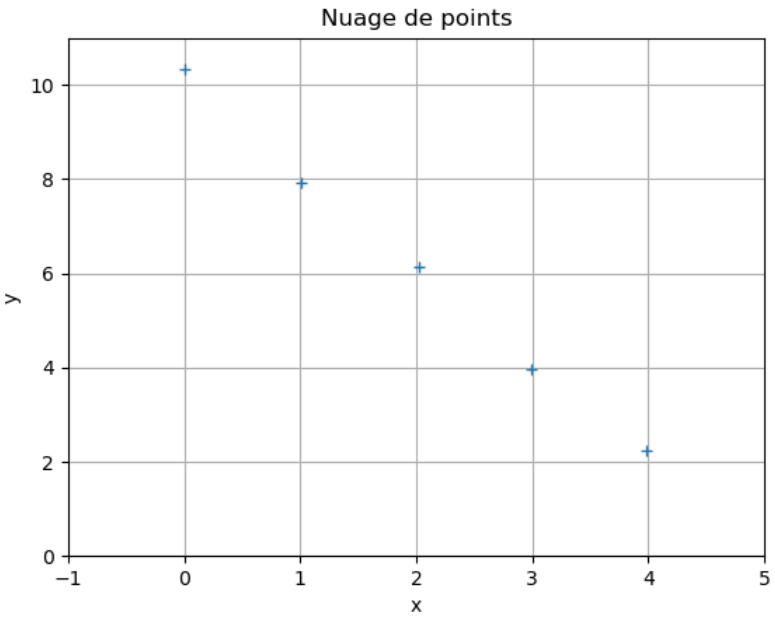
```
import matplotlib.pyplot as plt

x = [0, 1.01, 2.02, 2.99, 3.98]     # Données en abscisse
y = [10.32, 7.91, 6.13, 3.97, 2.25] # Données en ordonnée

plt.plot(x, y, '+')           # Tracé de la courbe
plt.title('Nuage de points')  # Ajout d'un titre
plt.xlabel('x')               # Nom de la grandeur en abscisse
plt.ylabel('y')               # Nom de la grandeur en ordonnée
plt.grid()                    # Ajout d'une grille
plt.show()                    # Affichage
```



— Le paramètre "+" dans la fonction `plot()` met en évidence les points avec un marqueur + sans les relier par des segments de droite.
— Les fonctions `title()`, `xlabel()` et `ylabel()` ajoutent une titre et les légendes sur les axes.
— La fonction `grid()` ajoute un quadrillage.

3.3.1.3 Définir une échelle

```
import matplotlib.pyplot as plt

x = [0, 1.01, 2.02, 2.99, 3.98]      # Données en abscisse
y = [10.32, 7.91, 6.13, 3.97, 2.25] # Données en ordonnée


plt.plot(x, y, 'x')      # Tracé de la courbe
plt.title('Mon titre')   # Ajout d'un titre
plt.xlabel('x')          # Nom de la grandeur en abscisse
plt.xlim(0,4)            # Echelle sur l'axe des x
plt.ylabel('y')          # Nom de la grandeur en ordonnée
plt.ylim(0,11)           # Echelle sur l'axe des y
plt.grid()               # Ajout d'une grille
plt.show()               # Affichage
```

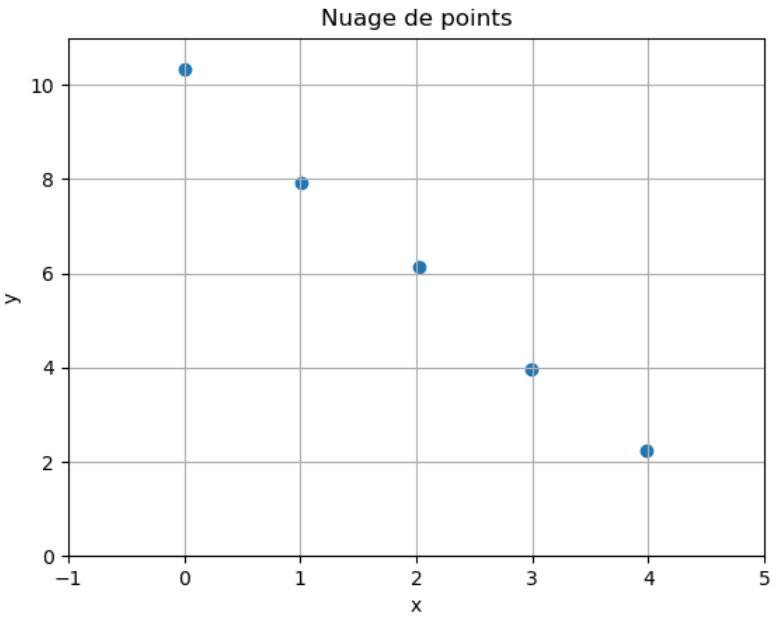


— Les fonctions `xlim()` et `ylim()` définissent les échelles respectivement sur l'axe des abscisses et l'axe des ordonnéées.

3.3.2 Fonction `scatter()`

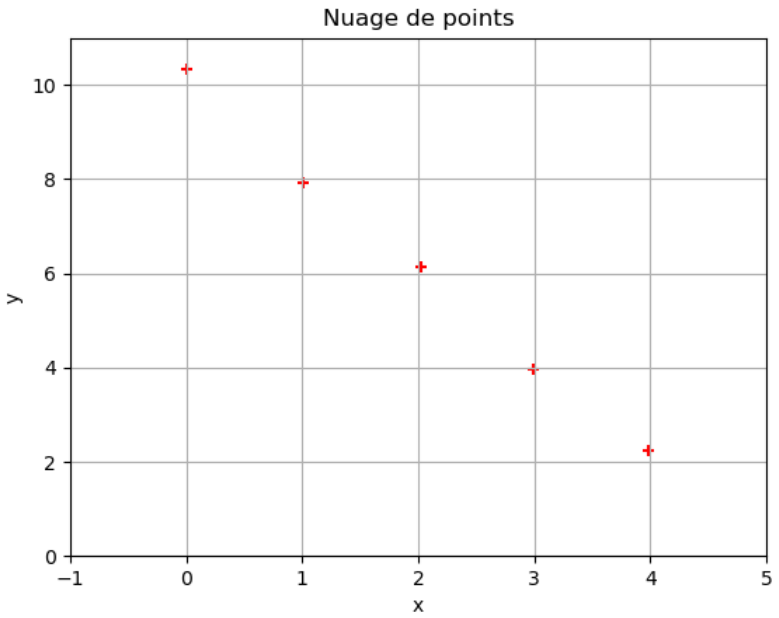
En réalité, la fonction `scatter()` est spécialement dédiée à la représentation d'un nuage de points. Mais cette dernière est assez rarement rencontrée au profit de la méthode `plot`.

```
import matplotlib.pyplot as plt

x = [0, 1.01, 2.02, 2.99, 3.98]      # Données en abscisse
y = [10.32, 7.91, 6.13, 3.97, 2.25] # Données en ordonnée

plt.scatter(x, y)              # Tracé de la courbe
plt.title('Nuage de points')   # Ajout d'un titre
plt.xlabel('x')                # Nom de la grandeur en abscisse
plt.xlim(-1,5)                 # Echelle sur l'axe des x
plt.ylabel('y')                # Nom de la grandeur en ordonnée
plt.ylim(0,11)                 # Echelle sur l'axe des y
plt.grid()                     # Ajout d'une grille
plt.show()                     # Affichage
```



Par défaut, la fonction `scatter()` effectue le tracé avec des gros points.

```
import matplotlib.pyplot as plt

x = [0, 1.01, 2.02, 2.99, 3.98]      # Données en abscisse
y = [10.32, 7.91, 6.13, 3.97, 2.25] # Données en ordonnée

plt.scatter(x, y, marker="+", color="red")  # Tracé de la courbe
plt.title('Nuage de points')                # Ajout d'un titre
plt.xlabel('x')                             # Nom de la grandeur en abscisse
plt.xlim(-1,5)                              # Echelle sur l'axe des x
plt.ylabel('y')                             # Nom de la grandeur en ordonnée
plt.ylim(0,11)                              # Echelle sur l'axe des y
plt.grid()                                  # Ajout d'une grille
plt.show()                                  # Affichage
```

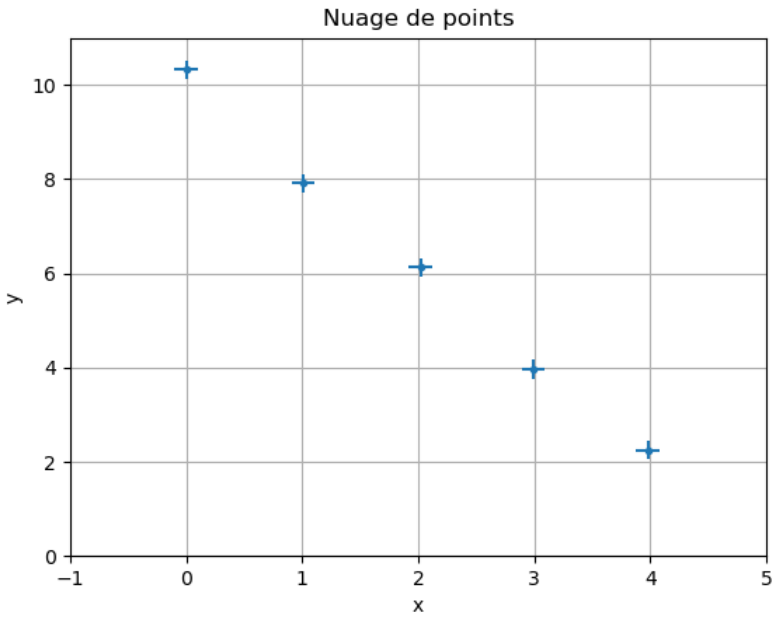


— L'option `marker` fixe le type de marqueur.
— L'option `color` définit la couleur des marqueur.

3.3.3 Fonction `errorbar()`

La fonction `errorbar()` trace un nuage de points avec des **barres d'incertitude** en x et en y.

```
import matplotlib.pyplot as plt

x = [0, 1.01, 2.02, 2.99, 3.98]      # Données en abscisse
y = [10.32, 7.91, 6.13, 3.97, 2.25] # Données en ordonnée
Ux = 0.10                            # Incertitude type de x
Uy = 0.20                            # Incertitude type de y

plt.errorbar(x, y, xerr=Ux, yerr=Uy, fmt=".") # Tracé de la courbe
plt.title('Nuage de points')                  # Ajout d'un titre
plt.xlabel('x')                               # Nom de la grandeur en abscisse
plt.xlim(-1,5)                                # Echelle sur l'axe des x
plt.ylabel('y')                               # Nom de la grandeur en ordonnée
plt.ylim(0,11)                                # Echelle sur l'axe des y
plt.grid()                                    # Ajout d'une grille
plt.show()                                    # Affichage
```



— Les options `xerr` et `yerr` définissent respectivement les incertitudes type pour les variables `x` et `y`.
— L'option `fmt = "."` (pour `format`) affiche uniquement les points.

3.4 Représentation graphique d'une fonction

3.4.1 Principe

Soit une fonction de la forme :

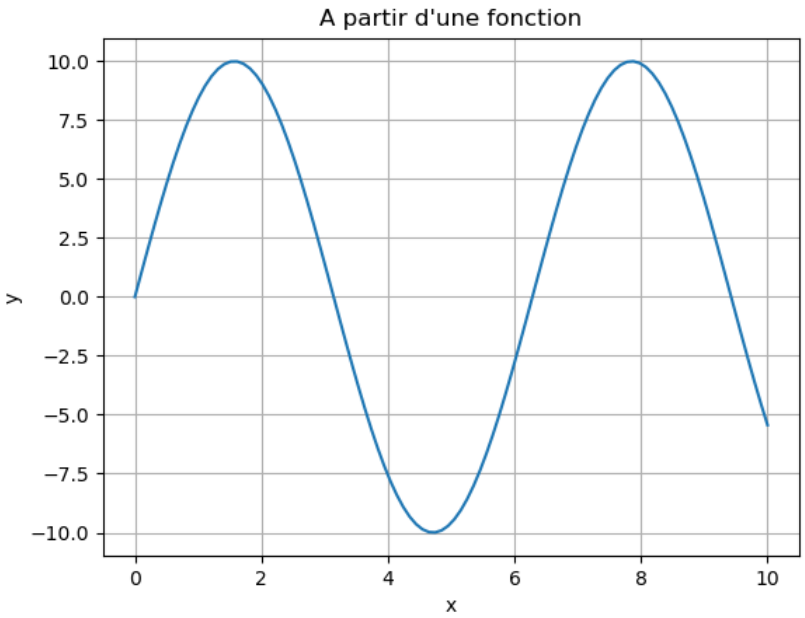
$$y = f(x)$$

Il est plus pratique de procéder comme suit :

— créer le tableau des `x` avec la fonction `linspace()` ou la fonction `arange()` de `Numpy`
— calculer le tableau des `y` à partir des fonctions mathématiques intégrées par `Numpy`.

3.4.2 Cas d'une seule fonction

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

x = np.linspace(0, 10, 100) # Création d'un tableau de valeurs pour x
y = 10*np.sin(x)            # Calcul de y à partir de la fonction mathématique

plt.plot(x, y)                       # Tracé de la courbe
plt.title("A partir d'une fonction") # Titre
plt.xlabel('x')                      # Légende abscisse
plt.ylabel('y')                      # Légende ordonnée
plt.grid()                           # Ajout d'une grille
plt.show()                           # Affichage
```

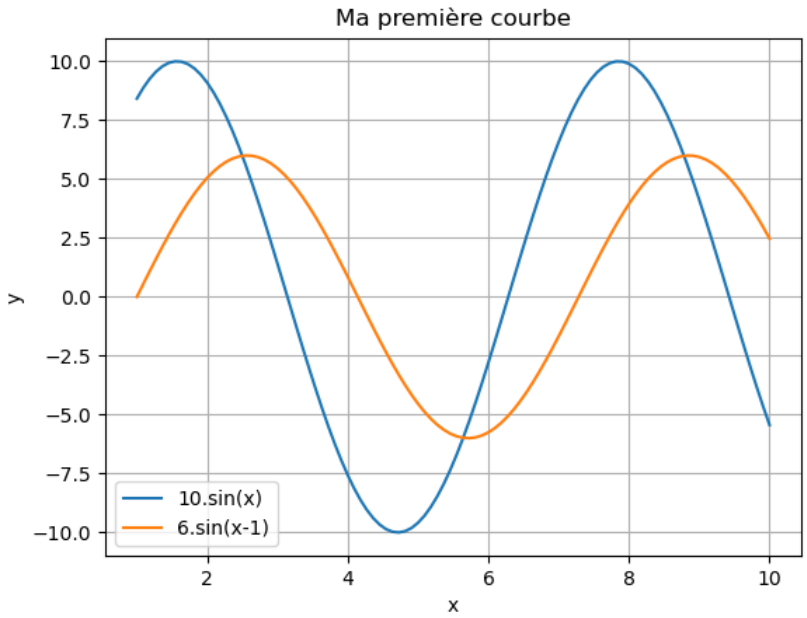



```
>>> x
array([ 0. ,  0.1010101 ,  0.2020202 , ... ,  9.7979798 ,  9.8989899 , 10.        ])
>>> y
array([ 0. ,  1.0083842 ,  2.00648857, ... , -3.64598734, -4.56637488, -5.44021111])
```

— La fonction `linspace(0,10,100)` génère un tableau Numpy de 100 éléments entre 0 à 10 inclus.

3.4.3 Cas de plusieurs fonctions

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

x = np.linspace(1, 10, 100) # Création d'un tableau de valeurs pour x
y1 = 10*np.sin(x)           # Calcul de y1
y2 =  6*np.sin(x-1)         # Calcul de y2

plt.plot(x, y1, label='10.sin(x)')   # Tracé de la courbe y1 avec texte légende
plt.plot(x, y2, label='6.sin(x-1)')  # Tracé de la courbe y1 avec texte légende
plt.title('Ma première courbe')      # Titre
plt.xlabel('x')                      # Légende abscisse
plt.ylabel('y')                      # Légende ordonnée
plt.legend()                         # Ajout de la légende
plt.grid()                           # Ajout d'une grille
plt.show()                           # Affichage
```



— Le paramètre `label` de la fonction `plot()` attribut une étiquette à la courbe.
— La fonction `legend()` affiche toutes les étiquettes dans une légende sur la figure.

3.5 Modélisation

3.5.1 Régression linéaire

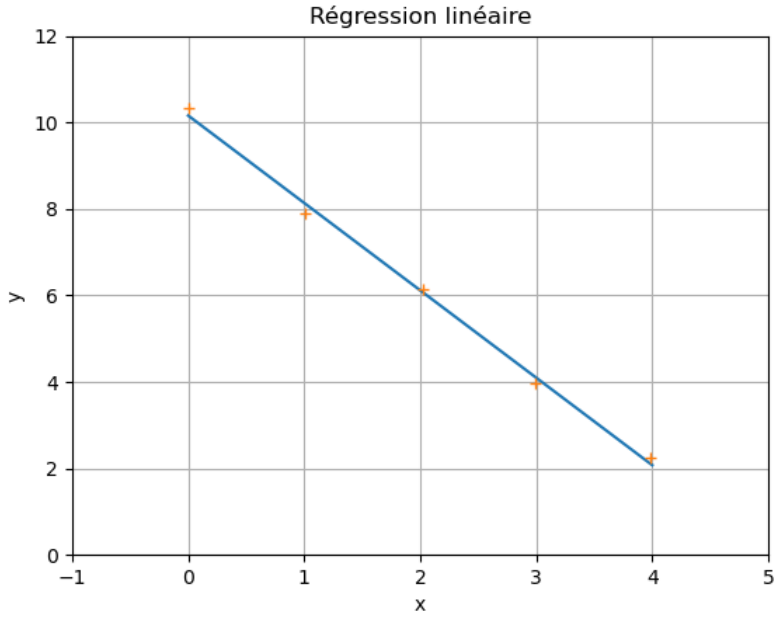
La fonction `linregress()` du module `scipy.stats` retourne les paramètres de la **régression linéaire** d'un nuage de points.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress

x = np.array([0, 1.01, 2.02, 2.99, 3.98])    # Données en abscisse
y = np.array([10.32, 7.91, 6.13, 3.97, 2.25]) # Données en ordonnée


a, b, rho ,_ ,_ = linregress(x, y)     # Régression linéaire
print("a = ", a)                       # Affichage de coefficient directeur
print("b = ", b)                       # Affichage de l'ordonnée à l'origine
print("rho = ", rho)                   # Affichage du coefficient de corrélation

x_mod = np.linspace(0, 4, 50)          # Nouvelle abscisse pour la modélisation
y_mod = a*x_mod + b                    # Ordonnées de la fonction affine

plt.plot(x_mod, y_mod, '-')            # Tracé de la droite
plt.plot(x, y, '+')                    # Tracé des points et de la fonction affine
plt.title('Régression linéaire')       # Titre
plt.xlabel('x')                        # Etiquette en abscisse
plt.xlim(-1,5)                         # Echelle en abscisse
plt.ylabel('y')                        # Etiquette en ordonnée
plt.ylim(0, 12)                        # Echelle en ordonnée
```

(suite sur la page suivante)

(suite de la page précédente)

```
plt.grid()
plt.show()                                  # Affichage
```




```
>>> %Run
a =  -2.0203420706406234
b =  10.156684141281247
rho =  -0.9986214342318739
```

— `a` est le coefficient directeur.
— `b` est l'ordonnée à l'origine.
— `rho` est le coefficient de corrélation linéaire (pas significatif en sciences physiques).

3.5.2 Modélisation à partir d'un polynôme

La fonction `polyfit()` du paquet `Numpy` retourne les paramètres d'un modèle polynomial d'un nuage de points.

3.5.2.1 Cas d'une parabole

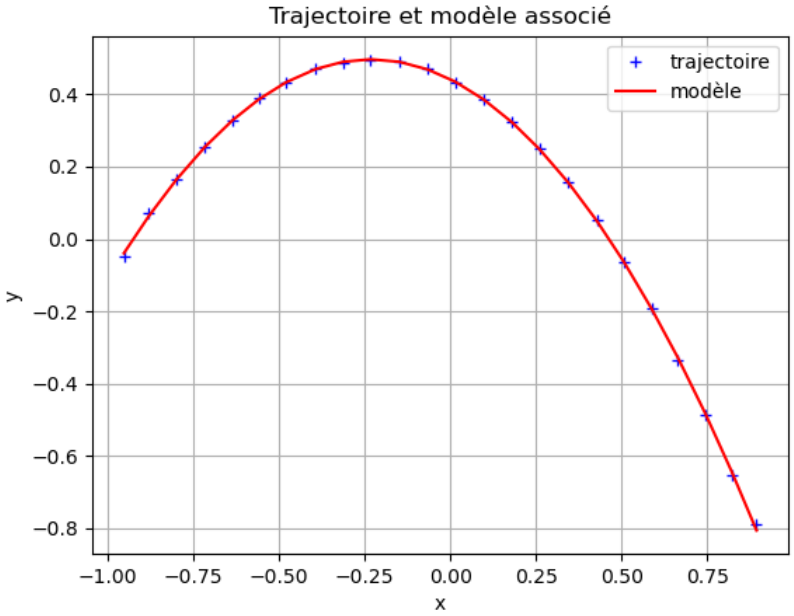
$$y = a \cdot x^2 + b \cdot x + c$$

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

# DONNEES
t = [0.0, 0.04, 0.08, 0.12, 0.16, 0.2, 0.24, 0.28, 0.32, 0.36, 0.4, 0.44, 0.48, 0.52, 0.
 ↪56, 0.6, 0.64, 0.68, 0.72, 0.76, 0.8, 0.84, 0.88, 0.92]
x = [-0.953328037081172, -0.879995111151852, -0.799995555592592, -0.716662685218364, -0.
 ↪636663129659105, -0.559996888914815, -0.479997333355555, -0.393331148166358, -0.
 ↪313331592607099, -0.233332037047839, -0.149999166673611, -0.066666296299383, 0.
 ↪013333259259877, 0.096666129634105, 0.179999000008333, 0.259998555567592, 0.
 ↪343331425941821, 0.426664296316049, 0.506663851875308, 0.586663407434568, 0.
 ↪663329648178858, 0.743329203738117, 0.819995444482407, 0.893328370411728]
```

(suite sur la page suivante)

(suite de la page précédente)

```
y = [-0.046666407409568, 0.069999611114352, 0.166665740748457, 0.253331925937654, 0.
↪326664851866975, 0.389997833351389, 0.433330925945988, 0.469997388910648, 0.
↪486663962985494, 0.493330592615432, 0.489997277800463, 0.469997388910648, 0.
↪433330925945988, 0.38666451853642, 0.323331537052006, 0.249998611122685, 0.
↪156665796303549, 0.053333037039506, -0.063332981484414, -0.189998944453241, -0.
↪333331481496913, -0.486663962985494, -0.65332970373395, -0.789995611147685]

# MODELISATION
a, b, c = np.polyfit(x, y, 2)  # Modélisation par un polynome de degré 2
print("a = ", a)
print("b = ", b)
print("c = ", c)

# CONSTRUCTION DU MODELE
x_mod = np.array(x)                # Création d'un tableau Numpy pour x
y_mod = a*x_mod**2 + b*x_mod + c   # Création d'un tableau Numpy pour y du modèle

# COURBES
plt.plot(x, y, "b+", label = "trajectoire")    # Courbe des mesures
plt.plot(x_mod, y_mod, 'r-', label = "modèle") # Courbe du modèle
plt.xlabel("x")                                # Etiquette en abscisse
plt.ylabel("y")                                # Etiquette en ordonnée
plt.title("Trajectoire et modèle associé")     # Titre
plt.legend()                                   # Affichage de la légende
plt.grid()                                     # Affichage de la grille
plt.show()                                     # Affichage de la figure
```

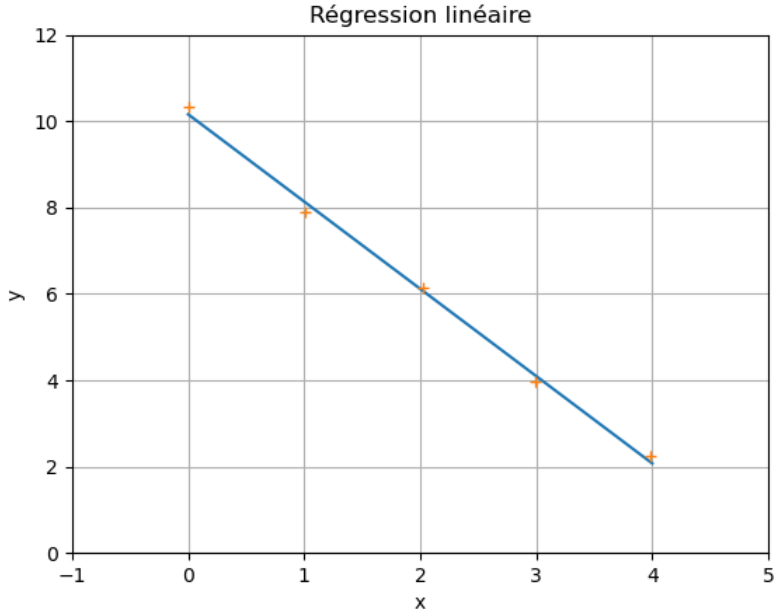



```
>>> %Run
a =  -1.028916427645116
b =  -0.47659343034449314
c =  0.4413962087861902
```

3.5.2.2 Cas d'une fonction affine (régression linéaire)

Il est également possible de procéder à une régression linéaire avec la fonction `polyfit()`.

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

x = np.array([0, 1.01, 2.02, 2.99, 3.98])    # Données en abscisse
y = np.array([10.32, 7.91, 6.13, 3.97, 2.25]) # Données en ordonnée

# MODELISATION
a, b = np.polyfit(x, y, 1)  # Modélisation par un polynome de degré 2
print("a = ", a)
print("b = ", b)

x_mod = np.linspace(0, 4, 50)          # Nouvelle abscisse pour la modélisation
y_mod = a*x_mod + b                    # Ordonnées de la fonction affine

plt.plot(x_mod, y_mod, '-')            # Tracé de la droite
plt.plot(x, y, '+')                    # Tracé des points et de la fonction affine
plt.title('Régression linéaire')       # Titre
plt.xlabel('x')                        # Etiquette en abscisse
plt.xlim(-1,5)                         # Echelle en abscisse
plt.ylabel('y')                        # Etiquette en ordonnée
plt.ylim(0, 12)                        # Echelle en ordonnée
plt.grid()
plt.show()                             # Affichage
```




```
>>> %Run
a =  -2.020342070640623
b =  10.156684141281245
```

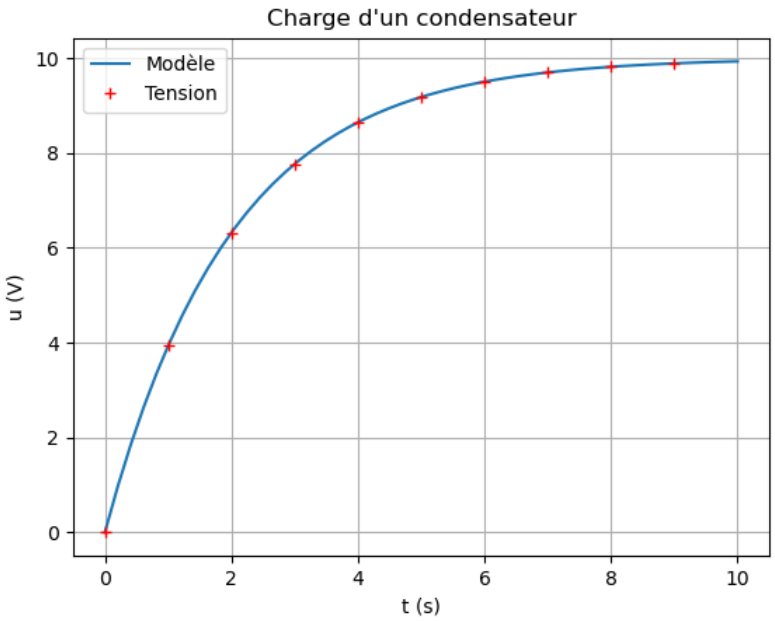
3.5.3 Modélisation à partir d'une fonction quelconque

La fonction `curve_fit()` du module `scipy.optimize` réalise un ajustement par rapport à une fonction mathématique quelconque et retourne les paramètres de cette fonction.

Considérons la charge d'un condensateur (initialement déchargé) de capacité C à travers un résistance R sous tension constante E :

$$u(t) = E \times (1 - e^{-t/\tau}) \qquad \text{avec} \qquad \tau = RC$$

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

# DONNEES
t = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
u = [0., 3.935, 6.321, 7.769, 8.647, 9.179, 9.502, 9.698, 9.817, 9.889]

# DEFINITION DE LA FONCTION QUELCONQUE
def fct(t, E, tau):
   return E*(1-np.exp(-t/tau))          # Expression du modèle

# MODELISATION
(E, tau), pcov = curve_fit(fct, t, u)   # Détermination des paramètres du modèle
print("E = ", E)                        # Affichage de A
print("tau =", tau)                     # Affichage de tau

# CONSTRUCTION DU MODELE
t_mod = np.linspace(0, 10, 50)
u_mod = fct(t_mod, E, tau)

# COURBES
plt.plot(t_mod, u_mod, label="Modèle")  # Courbe du modèle
plt.plot(t, u, "r+", label="Tension")   # Nuage des points
plt.title("Charge d'un condensateur")   # Titre
plt.xlabel("t (s)")                     # Etiquette en abscisse
plt.ylabel("u (V)")                     # Etiquette en ordonnée
plt.legend()                            # Affichage de la légende
plt.grid()                              # Affichage de la grille
plt.show()                              # Affichage de la figure
```

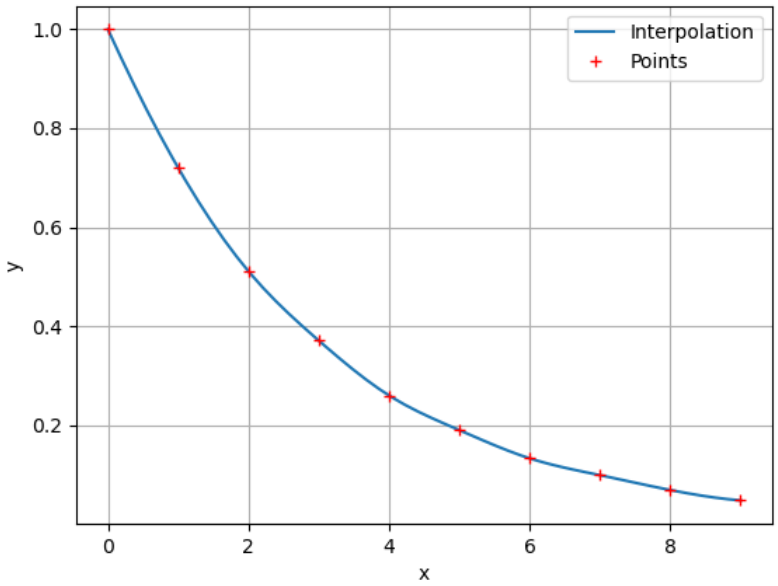



```
>>> %Run
E =  9.99999510282223
tau = 1.9999259182304618
```

3.5.4 Interpolation

La fonction `interp1d()` du module `scipy.interpolate` retourne une fonction Python pouvant estimer la position d'un point par interpolation.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

# DONNEES
x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
y = [1., 0.720, 0.511, 0.371, 0.260, 0.190, 0.133, 0.099, 0.069, 0.048]

# INTERPOLATION
f = interp1d(x, y, kind='cubic')  # ou kind = 'linear'

# CONSTRUCTION DE LA FONCTION
x_mod = np.linspace(0, 9, 50)     # Nouvelle abscisse pour f(x)
y_mod = f(x_mod)                  # Calcul des ordonnées de f(x)

# COURBES
plt.plot(x_mod, y_mod, label="Interpolation")  # Tracé de la fonction
plt.plot(x, y, 'r+', label="Points")           # Tracé des points
plt.xlabel("x")                                # Etiquette en abscisse
plt.ylabel("y")                                # Etiquette en ordonnée
plt.legend()                                   # Affichage de la légende
plt.grid()                                     # Affichage de la grille
plt.show()                                     # Affichage de la figure
```




```
>>> %Run
>>> f(1)
array(0.72)
>>> f(10)
ValueError: A value in x_new is above the interpolation range.
```

— La fonction `f()` est capable d'estimer par interpolation la position de n'importe quel point.
— L'interpolation n'est naturellement pas possible en dehors de la plage des valeurs de `x`.

```
>>> f.x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> f.y
array([1., 0.72, 0.511, 0.371, 0.26 , 0.19 , 0.133, 0.099, 0.069, 0.048])
```

— La fonction `f()` est bien définie par rapport au nuage des points passé en argument à la fonction `interp1d()`.

3.6 Tracer des vecteurs

???

Mouvement d'un point : vecteur vitesse (classe de seconde)

Mouvement d'un point : vecteur variation vitesse (classe de première)

3.7 Tracer un histogramme

...

Exercice d'application

Exercice 14. (histogramme et incertitude-type A)

Exemples au programme de physique-chimie

Histogramme d'une série de mesure (classe de terminale)

3.8 Calculs statistiques

3.8.1 Valeur moyenne et écart-type

Soit une série de valeurs numériques :

$$x_1, x_2, ..., x_n$$

Valeur moyenne :

$$\bar{x} = \frac{x_1 + x_2 + ... + x_n}{n}$$

Ecart-type :

$$s = \sqrt{\frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + ... + (x_n - \bar{x})^2}{n-1}}$$

3.8.2 Librairies Python pour le calcul statistique

Il est possible de réaliser des calculs statistiques avec le module `numpy` ou le module `satistics` (librairie standard).

Module	Fonction moyenne	Fonction écart-type
`numpy`	`mean(x)`	`std(x, ddof=1)`
`statistics`	`mean(x)`	`stdev(x)`

3.8.2.1 Module `numpy`

```
import numpy as np

# DONNEES
x = [10, 9, 15, 18, 11, 16, 15, 14, 16, 19, 12, 14, 15, 18]

# CACLULS
moy = np.mean(x)
s = np.std(x, ddof=1)

# AFFICHAGE
print("moyenne = ", moy)
print("ecart-type = ", s)
```

```
>>> %Run
moyenne =  14.428571428571429
ecart-type =  3.030975616334316
```

3.8.2.2 Module `statistics`

```
import statistics as stat

# DONNEES
x = [10, 9, 15, 18, 11, 16, 15, 14, 16, 19, 12, 14, 15, 18]

# CACLULS
moy = stat.mean(x)
s = stat.stdev(x)

# AFFICHAGE
print("moyenne = ", moy)
print("ecart-type = ", s)
```

```
>>> %Run
moyenne =  14.428571428571429
ecart-type =  3.030975616334316
```

Exercice d'application

Exercice 14. (histogramme et incertitude-type A)

3.9 Nombres aléatoires

3.9.1 Générer un entier aléatoire

3.9.2 Tirage d'un nombre aléatoire suivant une loi normale

3.10 Importer des données d'un fichier CSV

3.10.1 Qu'est-ce qu'un fichier CSV?

La plupart des logiciels de traitement de données (ex. tableur, Regressi, Latis, ...) offre la possibilité d'importer ou d'exporter des données dans un **fichier texte au format CSV (Comma Separated Values)** avec l'extension `.csv` ou `.txt`.

Le format CSV est un standard d'échange de données entre logiciels.

Note : L'édition d'un fichier texte (`.txt` ou `.csv`) se fait avec un **éditeur de texte** comme le logiciel **Notepad++**.

Pour illustrer la structure d'un fichier CSV, considérons les mesures de la caractéristique d'une résistance du tableau suivant.

Mesure	1	2	3	4	5	6	7	8	9	10
I (mA)	0	51	101	151	203	252	303	356	406	456
U (V)	0	1,11	2,22	3,28	4,42	5,5	6,68	7,73	8,92	9,91

L'édition des données de ce tableau dans un **fichier texte** nommé `data.txt` donne le contenu ci-dessous.

Code source 5 – Fichier `data.txt`

```
I,U
0,0
51,1.11
101,2.22
151,3.28
203,4.42
252,5.50
303,6.68
356,7.73
406,8.92
456,9.91
```

— Les **données sont toujours rangées en colonne**.
— La **première ligne renseigne sur les noms des variables**.
— Les **colonnes sont séparées par une virgule**.
— Le **séparateur décimal est un point** (comme en Python).

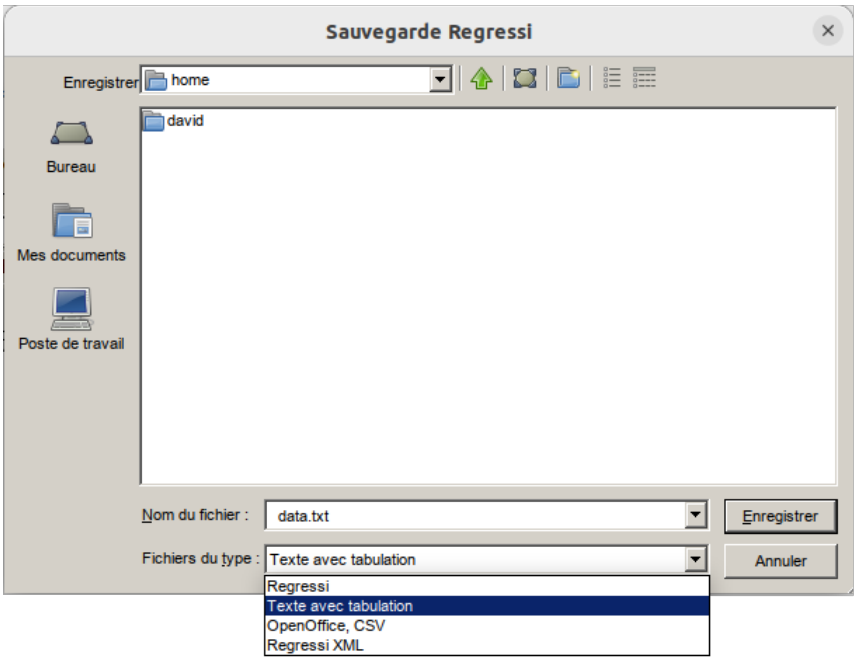
Des variantes du format CSV admettent d'autres symboles pour le séparateur de colonnes et le séparateur décimal.

Type	Séparateur de colonnes	Séparateur décimal
CSV (original)	,	.
CSV (européen)	; ou \t (tabulation)	,

Avertissement : Sachant que le langage Python impose le **point comme séparateur décimal**, le format CSV européen peut poser de problèmes de compatibilité.

3.10.2 Exportation en CSV de quelques logiciels

3.10.2.1 Regressi



Le logiciel Regressi propose une exportation des données en `.txt` ou `.csv` avec les règles suivantes :

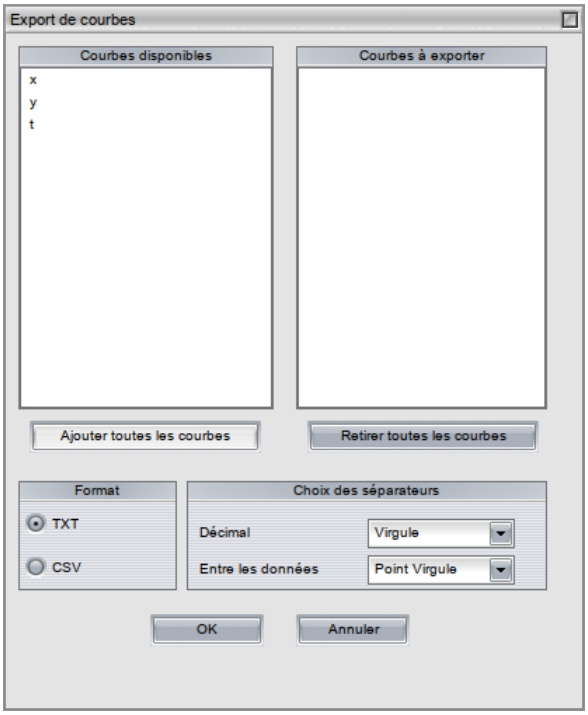
Format	Séparateur de colonnes	Séparateur décimal
Regressi (TXT)	\t	.
Regressi (CSV)	;	,

Code source 6 – Exemple de `data.txt` avec de Regressi.

```
I	U
mA	V
0	0
51	1.11
101	2.22
151	3.28
203	4.42
252	5.50
303	6.68
356	7.73
406	8.92
456	9.91
```

— Le **séparateur de colonnes** est la tabulation `\t`.
— Le **séparateur décimal** est le point `.`.
— Les **deux premières lignes ne sont pas à prendre en compte** (informations sur les données).

3.10.2.2 Latis Pro



Le logiciel Latis Pro supporte lui aussi les deux extensions `.txt` ou `.csv` pour l'exportation des données au format CSV.

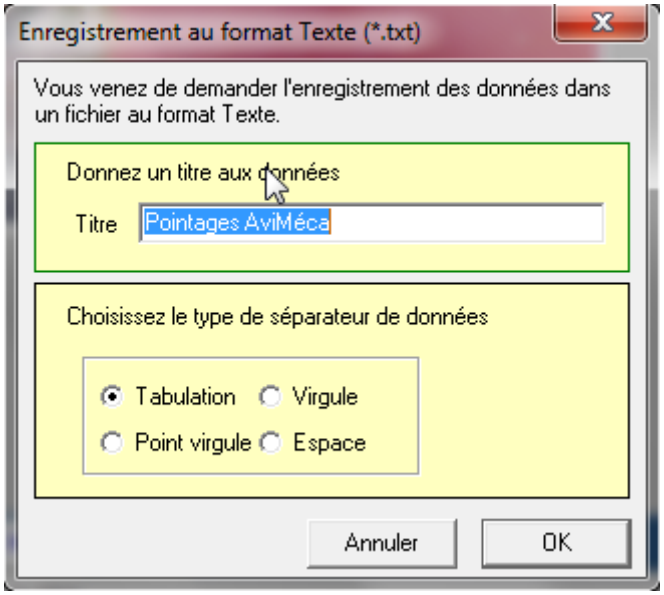
Format	Séparateur de colonnes	Séparateur décimal
Latis Pro (TXT)	; ou \t	, ou .
Latis Pro (CSV)	;	,

Code source 7 – Exemple de `data.txt` avec Latis Pro

```
I;U
0;0
51;1.11
101;2.22
151;3.28
203;4.42
252;5.50
303;6.68
356;7.73
406;8.92
456;9.91
```

— Le **séparateur de colonnes** est le point virgule `;`.
— Le **séparateur décimal** est le point `.`.
— La **première ligne n'est pas à prendre en compte** (informations sur les données).

3.10.2.3 AviMéca 3



Quant au logiciel AviMéca 3, seule l'extension `.txt` est disponible avec plusieurs séparateurs de colonnes et uniquement la virgule comme séparateur décimal.

Format	Séparateur de colonnes	Séparateur décimal
AviMéca (TXT)	\t ou , ou ;	,

Code source 8 – Exemple de `data.txt` avec AviMéca

```
Pointages AviMéca
t       x          y
(s)     (m)        (m)
0,000   4,18E-1   7,54E-1
0,033   4,85E-1   8,88E-1
0,067   5,52E-1   9,85E-1
0,100   6,12E-1   1,09E+0
0,133   6,79E-1   1,19E+0
0,167   7,54E-1   1,27E+0
0,200   8,21E-1   1,35E+0
```

— Le **séparateur de colonnes** est une tabulation `\t`.
— Le **séparateur décimal** est la virgule `,`.
— Les **trois premières lignes ne sont pas à prendre en compte** (informations sur les données).

3.10.3 Importation d'un fichier CSV avec le module `csv`

Le module `csv` est fourni par la librairie standard de Python.

3.10.4 Importation d'un fichier CSV avec le module `numpy`

Prenons l'exemple d'un fichier de données nommé `data.txt` dont le contenu est donné ci-dessous. Ce fichier texte est téléchargeable `ici`.

Code source 9 – Fichier `data.txt`

```
I;U
0;0
51;1.11
101;2.22
151;3.28
203;4.42
252;5.50
303;6.68
356;7.73
406;8.92
456;9.91
```

L'importation des données du fichier `data.txt` se fait assez simplement avec la fonction `loadtxt()` du paquet Numpy.

Avertissement : Le fichier `data.txt` doit-être placé dans le même répertoire que le programme Python.

```
import numpy as np
x, y = np.loadtxt('data.txt', delimiter=';', skiprows=1, unpack=True)
```

```
>>> %Run
>>> x
array([  0.,  51., 101., 151., 203., 252., 303., 356., 406., 456.])
>>> y
array([0.  , 1.11, 2.22, 3.28, 4.42, 5.5 , 6.68, 7.73, 8.92, 9.91])
```

— `loadtxt()` renvoie un tableau de tableaux (colonnes) Numpy.
— `delimiter=';'` signifie que le séparateur de colonnes est un point virgule.

— `skiprows=1` précise qu'il faut sauter la première ligne (pas de données).
— `unpack=True` transpose le tableau pour des données en colonne (en ligne par défaut).

Chapitre 4

Exercices d'application

Exercice 1. (dilution)

On veut préparer $V_f = 100$ mL d'une solution fille d'eau salé de concentration $C_f = 5\ \text{g}\cdot\text{L}^{-1}$ à partir d'une solution mère de concentration $C_m = 30\ \text{g}\cdot\text{L}^{-1}$.

Écrire un programme Python qui calcule :

— le facteur de dilution F ;
— le volume de la solution mère V_m à prélever ;
— le volume d'eau V_{eau} à compléter ;

et qui affiche un message de la forme suivante :

```
Prélever ?  mL de solution mère.
Puis ajouter ? mL d eau pour obtenir ? mL de solution fille.
Le facteur de dilution est de ?.
```

Exercice 2. (distance focale)

Le tableau ci-dessous donne plusieurs mesures de la distance focale d'une lentille.

f' (cm)	19,1	18,9	18,7	19,0	18,9	19,2	18,8	18,9

1. Mettre ce tableau dans une liste.
2. Comment obtenir la valeur de la première mesure ? De la dernière ?
3. Comment ajouter une nouvelle mesure égale à 19, 3 cm à la fin du tableau ?
4. Comment obtenir la **taille** de ce tableau ?
5. Comment obtenir la **moyenne**, la **valeur maximale** et la **valeur minimale** de cette série de mesures ?
6. Comment supprimer la **première valeur** et la **dernière valeur** de ce tableau ?

Exercice 3. (célérité son)

Compléter le code suivant qui doit calculer la moyenne de la liste `c`.

```
c = [335, 338, 341, 339, 340, 336, 343, 337, 341, 339, 337] # (m/s) célérité son

somme = 0

```

(suite sur la page suivante)

(suite de la page précédente)

```
for i in range(?):
    somme = somme + ?

moyenne = ?
print("Moyenne =", moyenne)
```

Exercice 4. (vitesse)

On donne les valeurs de l'altitude z d'une balle de tennis en chute libre en fonction du temps t.

Compléter le code ci-dessous qui doit calculer les valeurs prises par la vitesse v de cette balle pour chaque instant à partir de la relation :

$$v_n = \frac{z_{n+1} - z_{n-1}}{t_{n+1} - t_{n-1}} \qquad \text{ou} \qquad v_n = \frac{z_{n+1} - z_n}{t_{n+1} - t_n}$$

```
t = [0. , 0.04, 0.08, 0.12, 0.16, 0.20, 0.24, 0.28, 0.32, 0.36, 0.40, 0.44, 0.48] # (s)
z = [1.66, 1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21, 0. ] # (m)

v = []  # Initialisation d'une liste vide

for i in range(?):
    V = ?            # Calcul la vitesse en cours
    v.append(?)      # Mémorisation de la vitesse

print(v)
```

Exercice 5. (énergies)

On souhaite faire l'étude énergétique de cette même balle de tennis de masse $m = 55$ g.

Compléter le code ci-dessous afin de déterminer les valeurs prises par l'énergie potentielle de pesanteur E_{pp}, l'énergie cinétique E_C et l'énergie mécanique E_m pour chaque position de la balle.

```
z = [1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21] # (m)
v = [1.58, 1.95, 2.27, 2.69, 3.12, 3.49, 3.91, 4.38, 4.65, 4.91, 5.23] # (m/s)

Epp, Ec, Em = [], [], []  # Initialisations des tableaux

for i in range(?):
    EPP = ?        # Calcul l'énergie potentielle de pesanteur en cours
    EC = ?         # Calcul l'énergie cinétique en cours
    Epp.append(?) # Complète le tableau Epp
    Ec.append(?)  # Complète le tableau Ec
    Em.append(?)  # Complète le tableau Em

# Faire ici l'affichage
```

Exercice 6. (énergies)

Compléter le code ci-dessous qui reprend l'exercice précédent par **compréhension de liste**.

```
z = [1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21] # (m)
v = [1.58, 1.95, 2.27, 2.69, 3.12, 3.49, 3.91, 4.38, 4.65, 4.91, 5.23] # (m/s)

m = 55E-3  # (kg)   Masse de la balle
g = 9.81   # (m/s²) Accélération de la pesanteur

Epp = [? for Z in z]           # Tableau des énergies potentielles de pesanteur
Ec  = [?           ]           # Tableau des énergies cinétiques
Em  = [? for i in range(?)]  # Tableau des énergies mécaniques

# Faire ici l'affichage
```

Exercice 7. (nombre de dilutions)

Compléter le code ci-dessous qui calcule le **nombre de dilutions par 10** d'une solution mère de concentration C_m pour obtenir une solution fille de concentration C_f.

```
Cm = 1.0    # Concentration solution mère
Cf = 1.0e-3 # Concentration solution fille
Nb = ?    # Nombre de dilution

while ?:
    Nb = Nb + 1
    Cm = ?

print("Il faut réaliser", ?, "dilutions !")
```

Modifier le programme pour prendre en compte un facteur de dilution quelconque.

Exercice 8. (avancement)

Soit une réaction chimique totale décrite par l'équation suivante :

$$a \cdot A + b \cdot B = c \cdot C + d \cdot D$$

- A et B sont les réactifs
- C et D sont les produits.
- a, b, c, d sont les coefficients stœchiométriques.

Compléter le programme ci-dessous qui doit déterminer l'avancement final et toutes les quantités de matière à l'état final.

```
a, b, c, d = 1, 2, 3, 2                    # coefficients stoechiométiques
n_A, n_B, n_C, n_D = 0.5, 0.5, 0, 0        # (mol) Quantités de matière initiales

x = ?         # Initialisation de l'avancement
dx = ?        # Pas de l'avancement

while ? :
    x = x + dx      # Incrémentation de l'avancement
    n_A = ?
    n_B = ?
    n_C = ?
    n_D = ?
```

(suite sur la page suivante)

(suite de la page précédente)

```
# Faire ici l'affichage
```

Exercice 9. (dilution)

On considère la dilution de l'*exercice 1*.

1. Ecrire d'une fonction `dilution` qui renvoie le volume V_m de la solution mère à prélever à partir de C_m, C_f et V_f.
2. Améliorer cette fonction `dilution` pour qu'elle retourne le volume V_m, le volume d'eau V_{eau} à compléter et le facteur de dilution F.

— Ecrire une fonction `potentiel_hydrogene` qui calcule le pH d'une solution à partir de la relation :

$$pH = -\log(\frac{[H_3O^+]}{C_0}) \qquad \text{avec} \qquad C_0 = 1 \text{ mol} \cdot \text{L}^{-1}$$

— Ecrire également une fonction `concentration_ion_hydronium` qui fait l'opération inverse.

Exercice 11. (énergies)

Reprendre l'*exercice 5* en calculant les énergies E_{pp}, E_c et E_m à partir de tableaux Numpy des grandeurs z et v.

```
import numpy as np

z = [1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21] # (m)
v = [1.58, 1.95, 2.27, 2.69, 3.12, 3.49, 3.91, 4.38, 4.65, 4.91, 5.23] # (m/s)
```

Exercice 12. (positions et vecteurs vitesse)

Représenter sur un repère les **positions des points** de la balle de tennis de l'*exercice 4* lors de sa chute libre.

Faire apparaître également le **vecteur vitesse** pour chaque point. La composante en x du vecteur vitesse est considérée comme nulle !

```
import matplotlib.pyplot as plt

x = [.014, .014, .014, .009, .009, .005, .005, .001, .001, .001, .003] # (m)
z = [1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21] # (m)
v = [1.58, 1.95, 2.27, 2.69, 3.12, 3.49, 3.91, 4.38, 4.65, 4.91, 5.23] # (m/s)
```

Exercice 13. (positions et vecteurs vitesse)

Faire le même travail pour un lancé d'une balle. Les composantes v_x et v_y sont à calculer.

```
t = [0.0, 0.0667, 0.1334, 0.2001, 0.2668, 0.3335, 0.4002, 0.4669, 0.5336, 0.6003, 0.667,
 ↪ 0.7337, 0.8004, 0.8671, 0.9338]
x = [0.003, 0.141, 0.275, 0.410, 0.554, 0.686, 0.820, 0.958, 1.089, 1.227, 1.359, 1.490,
 ↪ 1.599, 1.705, 1.801]
y = [0.746, 0.990, 1.175, 1.336, 1.432, 1.505, 1.528, 1.505, 1.454, 1.355, 1.207, 1.018,
 ↪ 0.797, 0.544, 0.266]
```

Exercice 14. (histogramme et incertitude-type A)

Reprendre la répétition des mesures de la distance focale d'une lentille de l'*exercice 2*.

```
f = [19.1, 18.9, 18.7, 19.0, 18.9, 19.2, 18.8, 18.7] # (mm)
```

1. Tracer l'histogramme de cette série de mesures.
2. Déterminer la valeur moyenne, l'écart-type et l'évaluation de l'incertitude du type A.
3. Afficher le résultat de la mesure sous la forme suivante :

$$X = \bar{x} \pm u(\bar{x}) \text{ unité}$$

Chapitre 5

Nouveaux programmes du lycée

5.1 Classe de seconde générale et technologique

5.1.1 Caractéristique d'un dipôle

Programme de seconde générale et technologique 2019.

"Représenter un nuage de points associé à la caractéristique d'un dipôle et modéliser la caractéristique de ce dipôle à l'aide d'un langage de programmation".

5.1.1.1 Principe

— Tracer le nuage de points des mesures de la tension et de l'intensité du courant d'un dipôle électrique.
— Modéliser cette caractéristique quand cela est possible.

5.1.1.2 Exemple 1 : caractéristique d'une résistance

Les mesures de l'intensité du courant I et de la tension U aux bornes d'une résistance de 220 Ω à 5% sont données dans le tableau ci-dessous.

Mesure	1	2	3	4	5	6	7	8	9	10
I (mA)	0	51	101	151	203	252	303	356	406	456
U (V)	0	1,11	2,22	3,28	4,42	5,5	6,68	7,73	8,92	9,91

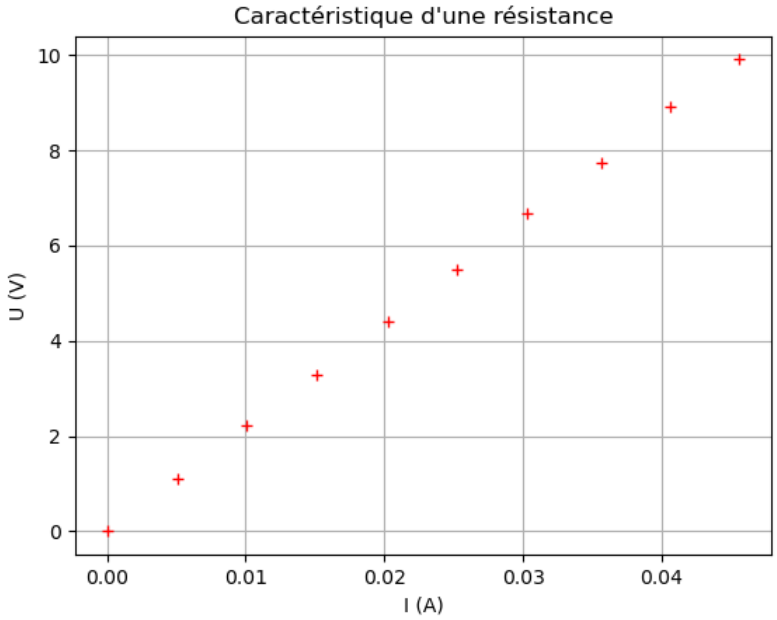
Tracé du nuage de points

```
import matplotlib.pyplot as plt

# MESURES
I = [0., 0.0051, 0.0101, 0.0151, 0.0203, 0.0252, 0.0303, 0.0356, 0.0406, 0.0456]
U = [0., 1.11  , 2.22  , 3.28  , 4.42  , 5.5   , 6.68  , 7.73  , 8.92  , 9.91  ]

# CARACTERISTIQUE
plt.plot(I, U, "r+")
plt.title("Caractéristique d'une résistance")
plt.xlabel("I (A)")
plt.ylabel("U (V)")
```

(suite sur la page suivante)

(suite de la page précédente)

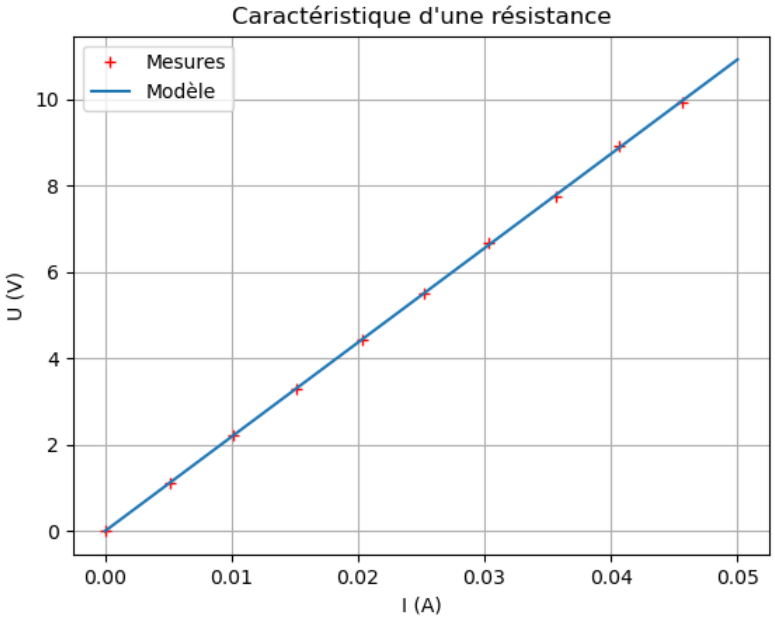
```
plt.grid()
plt.show()
```



Modélisation avec `scipy.stats.linregress()` Régression linéaire par la méthode des moindres carrés.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress

# MESURES
I = [0., 0.0051, 0.0101, 0.0151, 0.0203, 0.0252, 0.0303, 0.0356, 0.0406, 0.0456]
U = [0., 1.11  , 2.22  , 3.28  , 4.42  , 5.5   , 6.68  , 7.73  , 8.92  , 9.91  ]

# MODELISATION
a, b, _, _, _ = linregress(I, U)
print("a =", a)
print("b =", b)
print("R =", round(a), "Ohm")

# CONSCTRUCTION DU MODELE
I_mod = np.array([0, 0.05])
U_mod = a*I_mod + b

# COURBE
plt.plot(I, U, "r+", label="Mesures")
plt.plot(I_mod, U_mod, label="Modèle")
plt.legend()
plt.title("Caractéristique d'une résistance")
plt.xlabel("I (A)")
plt.ylabel("U (V)")
plt.grid()
plt.show()
```

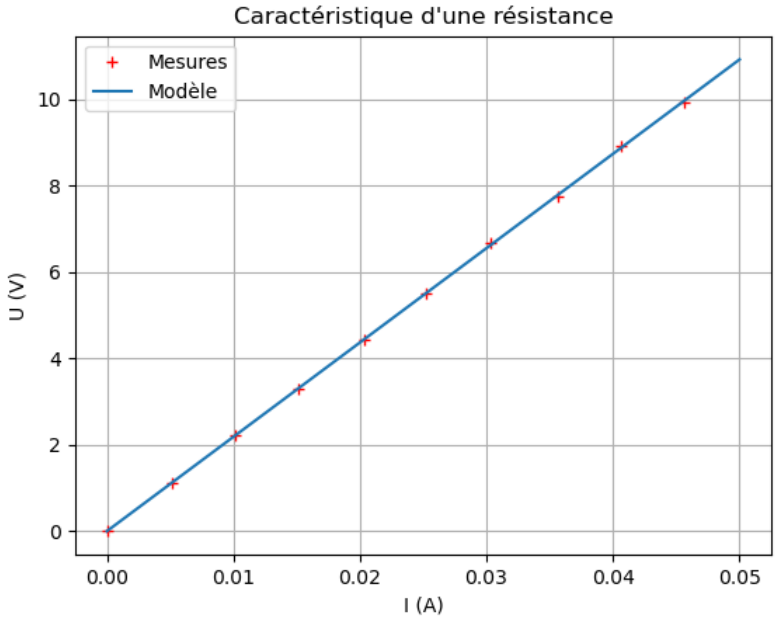
```
>>> %Run
a = 218.29027146901586
b = 0.0021647132211279896
R = 218 Ohm
```



Modélisation avec `numpy.polyfit()` La régression linéaire avec `numpy.polyfit()` donne le même résultat.

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

# MESURES
I = [0., 0.0051, 0.0101, 0.0151, 0.0203, 0.0252, 0.0303, 0.0356, 0.0406, 0.0456]
U = [0., 1.11  , 2.22  , 3.28  , 4.42  , 5.5   , 6.68  , 7.73  , 8.92  , 9.91  ]

# MODELISATION
a, b = np.polyfit(I, U, 1)
print("a =", a)
print("b =", b)
print("R =", round(a), "Ohm")

# CONSCTRUCTION DU MODELE
I_mod = np.array([0, 0.05])
U_mod = a*I_mod + b

# COURBE
plt.plot(I, U, "r+", label="Mesures")
plt.plot(I_mod, U_mod, label="Modèle")
plt.legend()
plt.title("Caractéristique d'une résistance")
plt.xlabel("I (A)")
plt.ylabel("U (V)")
plt.grid()
plt.show()
```

```
>>> %Run
a = 218.29027146901592
b = 0.002164713221130428
R = 218 Ohm
```



Modélisation avec `scipy.optimize.curve_fit()` Une modélisation à partir d'une **fonction linéaire** est plus adaptée au nuage de points obtenu.

$$f(x) = a \times x$$

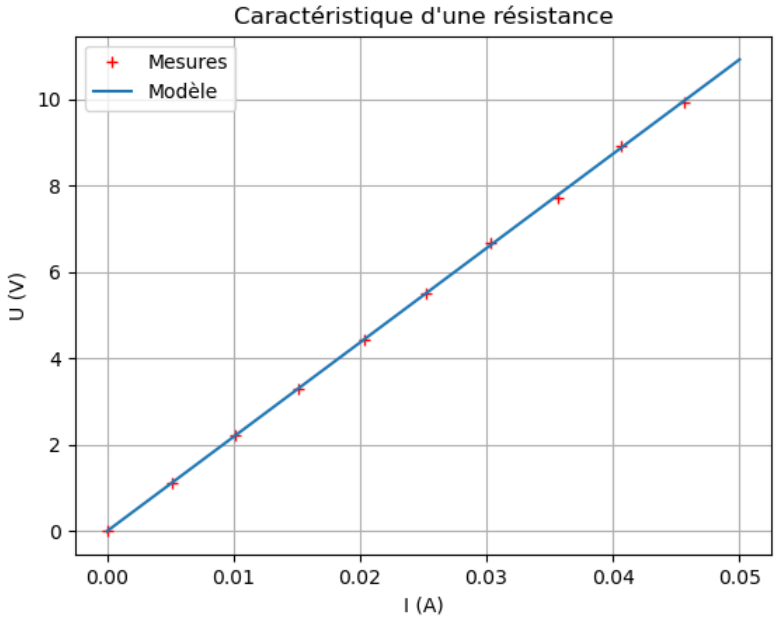
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# MESURES
I = [0., 0.0051, 0.0101, 0.0151, 0.0203, 0.0252, 0.0303, 0.0356, 0.0406, 0.0456]
U = [0., 1.11  , 2.22  , 3.28  , 4.42  , 5.5   , 6.68  , 7.73  , 8.92  , 9.91  ]

# DEFINITION DE LA FONCTION
def fct(x, a):
      return a*x          # Expression du modèle

# MODELISATION
(a), _ = curve_fit(fct, I, U)    # Détermination des paramètres du modèle
print("a =", a)
print("R =", round(a[0]), "Ohm")

# CONSCTRUCTION DU MODELE
I_mod = np.array([0, 0.05])
U_mod = a*I_mod

# COURBE
plt.plot(I, U, "r+", label="Mesures")
```

(suite sur la page suivante)

(suite de la page précédente)

```
plt.plot(I_mod, U_mod, label="Modèle")
plt.legend()
plt.title("Caractéristique d'une résistance")
plt.xlabel("I (A)")
plt.ylabel("U (V)")
plt.grid()
plt.show()
```

```
>>> %Run
a = [218.35770119]
R = 218 Ohm
```



5.1.1.3 Exemple 2 : caractéristique d'une CTN

Nous considérons maintenant un capteur non linéaire. Il s'agit d'un CTN de 10K du type EKS 221. Le tableau ci-dessous donne les mesures de la température en fonction de la résistance électrique de ce capteur.

Mesure	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T (°C)	2	6	10	15	20	25	30	35	40	45	50.1	55	60	65	70	75	80	85
R (Ohm)	25378	21487	18301	14990	12402	10295	8615	7226	6097	5121	4306	3632	3070	2609	2221	1903	1630	1404

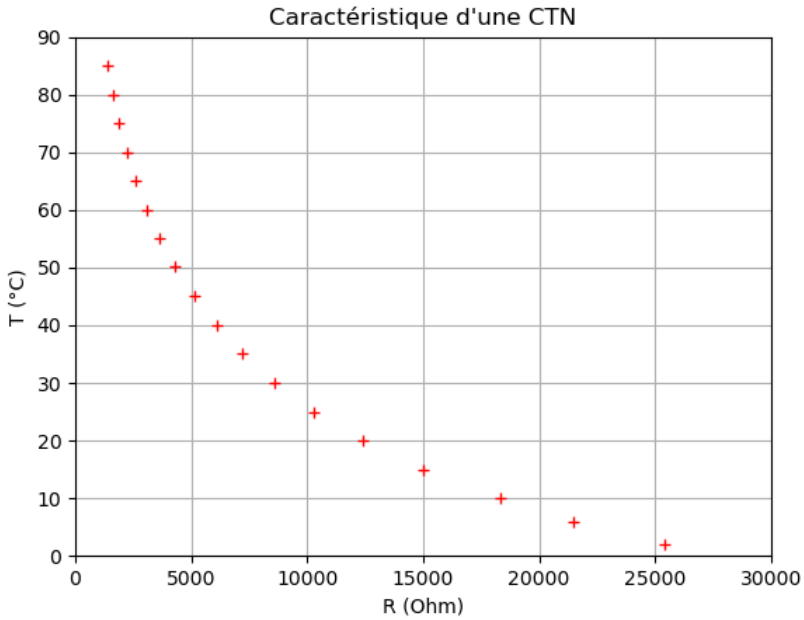
Nuage de points

```
import matplotlib.pyplot as plt

# DONNEES
T = [2, 6, 10, 15, 20, 25, 30, 35, 40, 45, 50.1, 55, 60, 65, 70, 75, 80, 85]
R = [25378,21487,18301,14990,12402,10295,8615,7226,6097,5121,4306,3632,3070,2609,2221,
 ↪1903,1630,1404]
```

(suite sur la page suivante)

(suite de la page précédente)

```
# COURBE
plt.plot(R, T, "r+")                        # Tracé du nuage de points en rouge
plt.title("Caractéristique d'une CTN")      # Ajout d'un titre
plt.xlabel("R (Ohm)")                       # Ajout d'une légende sur l'abscisse
plt.xlim(0, 30000)                          # Echelle en abscisse
plt.ylabel("T (°C)")                        # Ajout d'une légende sur l'ordonnée
plt.ylim(0, 90)                             # Echelle en ordonnée
plt.grid()                                  # Ajoute une grille
plt.show()                                  # Affiche la figure
```



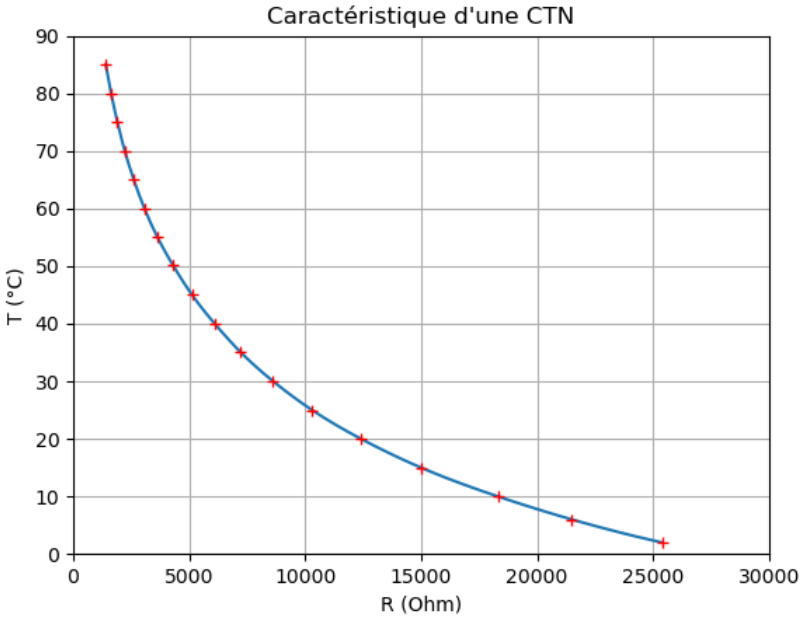
Modélisation par une interpolation avec `scipy.interpolate.interp1d()` Il est possible de modéliser la caractéristique de ce capteur par une interpolation et obtenir ainsi une fonction d'étalonnage de la CTN.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

# DONNEES
T = [2, 6, 10, 15, 20, 25, 30, 35, 40, 45, 50.1, 55, 60, 65, 70, 75, 80, 85]
R = [25378,21487,18301,14990,12402,10295,8615,7226,6097,5121,4306,3632,3070,2609,2221,
 ↪1903,1630,1404]

# INTERPOLATION
f = interp1d(R, T, kind='cubic')

# CONSTRUCTION DU MODELE
R_int = np.linspace(1404, 25378, 100)
T_int = f(R_int)

# COURBE
plt.plot(R_int, T_int)
plt.plot(R, T, "r+")                        # Tracé du nuage de points en rouge
```

(suite sur la page suivante)

(suite de la page précédente)

```
plt.title("Caractéristique d'une CTN")  # Ajout d'un titre
plt.xlabel("R (Ohm)")                   # Ajout d'une légende sur l'abscisse
plt.xlim(0, 30000)                      # Echelle en abscisse
plt.ylabel("T (°C)")                    # Ajout d'une légende sur l'ordonnée
plt.ylim(0, 90)                         # Echelle en ordonnée
plt.grid()                              # Ajoute une grille
plt.show()                              # Affiche la figure
```



Utilisation de la courbe d'étalonnage ~~~~~~éééé

Il est intéressant de noter que la fonction `f()` retourne par interpolation la valeur de la température du capteur pour n'importe quelle valeur de la résistance dans la plage de mesure.

Le programme suivant donne la température à partir d'une mesure à l'ohmmètre de la résistance de la CTN.

```
from scipy.interpolate import interp1d

# DONNEES
T = [2, 6, 10, 15, 20, 25, 30, 35, 40, 45, 50.1, 55, 60, 65, 70, 75, 80, 85]
R = [25378,21487,18301,14990,12402,10295,8615,7226,6097,5121,4306,3632,3070,2609,2221,
 ↪1903,1630,1404]

# INTERPOLATION
f = interp1d(R, T, kind='cubic')

# SAISIE ET CALCUL
while True:
   res = float(input("R (Ohm) = "))  # Saisie de la résistance
   if res == 0:                      #
      break                          # Stop de la boucle pour R = 0
   temp = f(res)                     # Caclul de la température
   print("T (°C) =", temp)           # Affichage

print("Fin !")
```

```
>>> %Run
R (Ohm) = 10000
T (°C) = 25.806756762710716
R (Ohm) = 15000
T (°C) = 14.982769995915337
R (Ohm) = 20000
T (°C) = 7.788406834764948
R (Ohm) = 0
Fin !
```

Note : Il est envisageable de mesurer la résistance de la CTN automatiquement par un micro-contrôleur programmé en Python et d'en déduire la température.

5.1.2 Mouvement d'un point : positions

Programme de seconde générale et technologique 2019.

"Représenter les positions successives d'un système modélisé par un point lors d'une évolution unidimensionnelle ou bidimensionnelle à l'aide d'un langage de programmation".

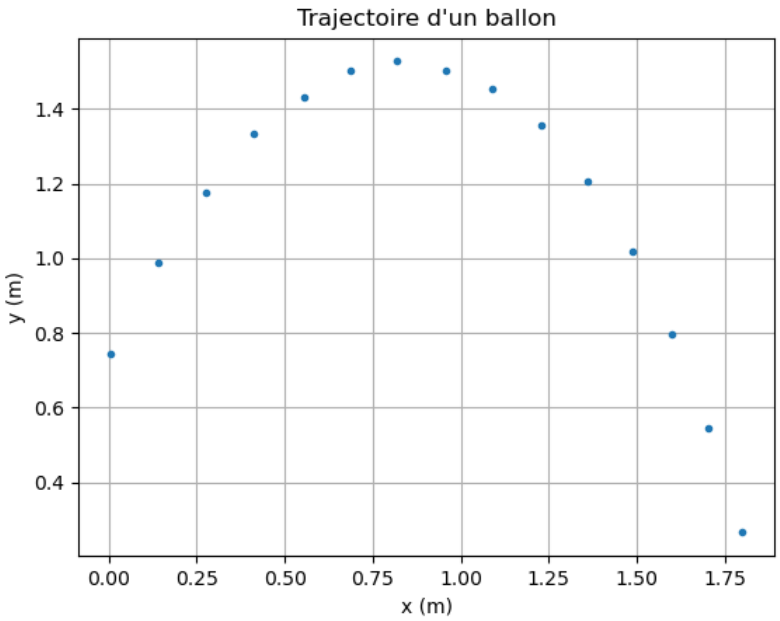
5.1.2.1 Principe

Représenter un nuage de points à partir des positions d'un système modélisé par un point.

5.1.2.2 Exemple : lancé de ballon

Les données suivantes sont obtenues à partir de l'exploitation d'une chronophotographie du lancé d'un ballon.

```
import matplotlib.pyplot as plt

# DONNEES
x = [0.003, 0.141, 0.275, 0.410, 0.554, 0.686, 0.820, 0.958, 1.089, 1.227, 1.359, 1.490,
 ↪ 1.599, 1.705, 1.801]
y = [0.746, 0.990, 1.175, 1.336, 1.432, 1.505, 1.528, 1.505, 1.454, 1.355, 1.207, 1.018,
 ↪ 0.797, 0.544, 0.266]

# COURBE
plt.plot(x, y, ".")
plt.title("Trajectoire d'un ballon")
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.grid()
plt.show()
```



5.1.3 Mouvement d'un point : vecteur vitesse

Programme de seconde générale et technologique 2019.

"Représenter des vecteurs vitesse d'un système modélisé par un point lors d'un mouvement à l'aide d'un langage de programmation".

5.1.3.1 Principe

— Calculer les coordonnées (v_{x_i}, v_{y_i}) des composantes des vecteurs vitesse à partir des relations :

$$v_{x_i} = \frac{x_{i+1} - x_{i-1}}{t_{i+1} - t_{i-1}} = \frac{x_{i+1} - x_{i-1}}{2\Delta t} \qquad \text{et} \qquad y_{x_i} = \frac{y_{i+1} - y_{i-1}}{t_{i+1} - t_{i-1}} = \frac{y_{i+1} - y_{i-1}}{2\Delta t}$$

— Dessiner les vecteurs vitesse correspondant sur la figure.

5.1.3.2 Exemple : lancé d'un ballon

Les données sont obtenues à partir de l'exploitation d'une chronophotographie du lancé d'un ballon.

Le programme ci-dessous calcule les composantes `vx` et `vy` des vecteurs vitesse dans un boucle `for` puis trace ces vecteurs sur la figure avec la fonction `quiver()`.

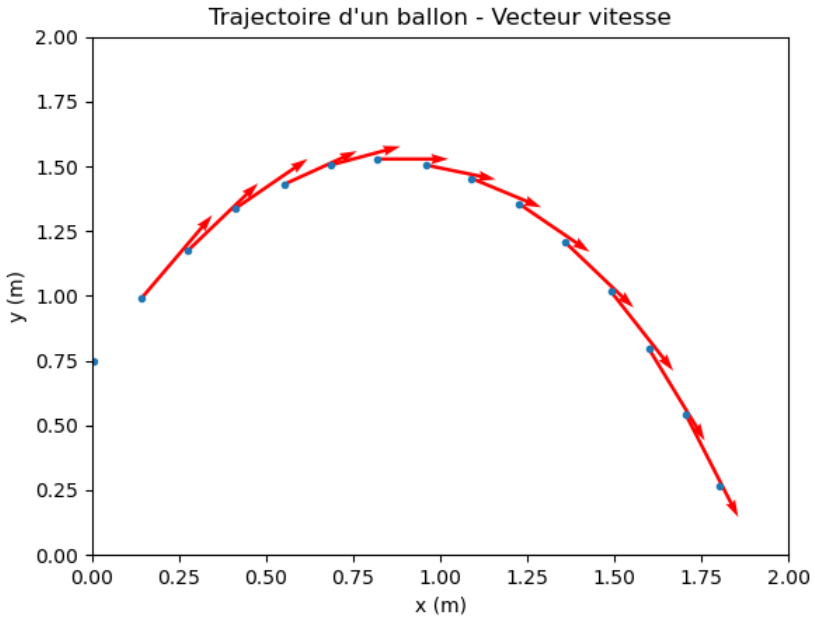
Calcul et tracé des vecteurs vitesse

```
import matplotlib.pyplot as plt

# DONNEES
dt = 0.0667
x = [0.003, 0.141, 0.275, 0.410, 0.554, 0.686, 0.820, 0.958, 1.089, 1.227, 1.359, 1.490,
 ↪ 1.599, 1.705, 1.801]
y = [0.746, 0.990, 1.175, 1.336, 1.432, 1.505, 1.528, 1.505, 1.454, 1.355, 1.207, 1.018,
 ↪ 0.797, 0.544, 0.266]
```

(suite sur la page suivante)

(suite de la page précédente)

```
# CALCULS DES COMPONSANTES DES VITESSES
N = len(x)   # Nombre de points de mesures
vx = [0]*N   # Liste de N éléments à zéro
vy = [0]*N   # Liste de N éléments à zéro
for i in range(1, N-1):
   vx[i] = (x[i+1]-x[i-1])/(2*dt)
   vy[i] = (y[i+1]-y[i-1])/(2*dt)

# COURBES
plt.plot(x, y, ".")                    # Affichage des points
for i in range(N):
   plt.quiver(x[i], y[i], vx[i], vy[i], angles='xy', scale_units='xy', scale=10, color=
 ↪'red', width=0.005)
plt.title("Trajectoire d'un ballon - Vecteur vitesse")  # Titre
plt.xlabel("x (m)")                    # Légende en x
plt.ylabel("y (m)")                    # Légende en y
plt.show()
```



Amélioration et affichage des vitesses Dans le programme précédent, les listes `vx` et `vy` ont été initialisées par des zéros. **En toute rigueur, les premières valeurs et les dernières valeurs de ces listes ne sont pas définies et ne peuvent pas être nulle.** Le langage Python propose la valeur particulière `nan` (Not A Number) pour des situations comme celle là.

L'affichage passe par le calcul des vitesses avec la relation :

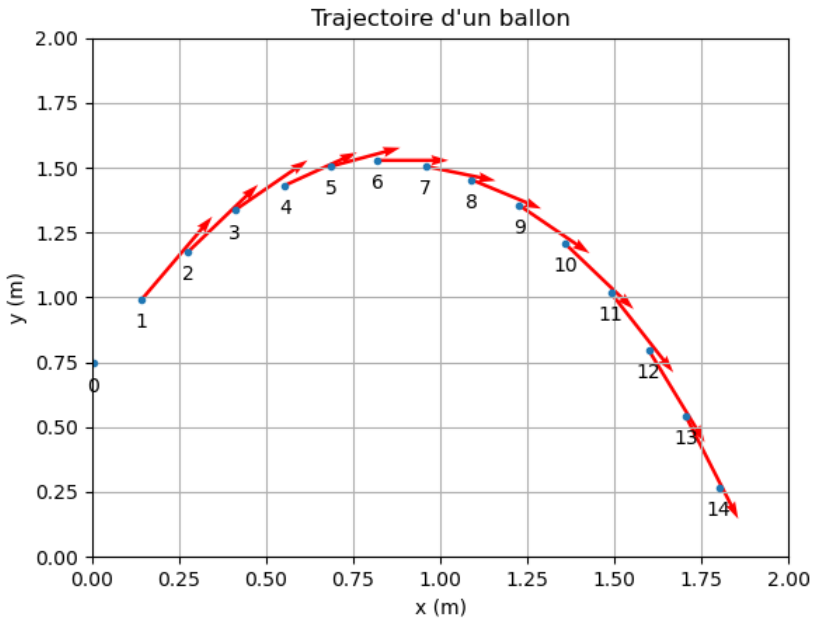
$$v_i = \sqrt{v_{x_i}^2 + v_{y_i}^2}$$

```
import matplotlib.pyplot as plt
from math import nan, sqrt

# DONNEES
dt = 0.0667
```

(suite sur la page suivante)

(suite de la page précédente)

```
x = [0.003, 0.141, 0.275, 0.410, 0.554, 0.686, 0.820, 0.958, 1.089, 1.227, 1.359, 1.490,
 ↪ 1.599, 1.705, 1.801]
y = [0.746, 0.990, 1.175, 1.336, 1.432, 1.505, 1.528, 1.505, 1.454, 1.355, 1.207, 1.018,
 ↪ 0.797, 0.544, 0.266]

# CALCULS DES COMPONSANTES DES VITESSES
N = len(x)      # Nombre de points de mesures
vx = [nan]*N    # Liste de N éléments NAN (Not A Number)
vy = [nan]*N    # Liste de N éléments NAN (Not A Number)
for i in range(1, N-1):
    vx[i] = (x[i+1]-x[i-1])/(2*dt)
    vy[i] = (y[i+1]-y[i-1])/(2*dt)

# CALCULS DES VITESSES
v = [sqrt(vx[i]**2+vy[i]**2) for i in range(N)]
for i in range(N):
    print("v_{:2} = {:.2f} m/s".format(i, round(v[i],2)))

# COURBES
plt.plot(x, y, ".")                    # Affichage des points
for i in range(N):
    plt.quiver(x[i], y[i], vx[i], vy[i], angles='xy', scale_units='xy', scale=10, color=
 ↪'red', width=0.005)
    plt.text(x[i], y[i]-0.05, i, va="top", ha="center")
plt.title("Trajectoire d'un ballon")  # Titre
plt.xlabel("x (m)")                    # Légende en x
plt.ylabel("y (m)")                    # Légende en y
plt.xlim(0, 2)                         # Echelle en x
plt.ylim(0, 2)                         # Echelle en y
plt.grid()
plt.show()
```




```
>>> %Run
v_ 0 = nan m/s
```

(suite sur la page suivante)

(suite de la page précédente)

```
v_ 1 = 3.81 m/s
v_ 2 = 3.29 m/s
v_ 3 = 2.84 m/s
v_ 4 = 2.43 m/s
v_ 5 = 2.12 m/s
v_ 6 = 2.04 m/s
v_ 7 = 2.09 m/s
v_ 8 = 2.31 m/s
v_ 9 = 2.74 m/s
v_10 = 3.20 m/s
v_11 = 3.56 m/s
v_12 = 3.90 m/s
v_13 = 4.26 m/s
v_14 = nan m/s
```

— Dans cet exemple, la valeur `nan` est donnée par le module `math`. Le paquet `numpy` définit également une valeur `nan`. Sinon il faudrait écrire `float("nan")`.
— Dans la fonction `print()` la méthode `format()` est plus pratique pour la mise en forme de l'affichage.

5.2 Classe première, enseignement de spécialité

5.2.1 Mouvement d'un point : vecteur variation vitesse

Programme de première générale - Enseignement de spécialité - 2019

"Utiliser un langage de programmation pour étudier la relation approchée entre la variation du vecteur vitesse d'un système modélisé par un point matériel entre deux instants voisins et la somme des forces appliquées sur celui-ci".

5.2.1.1 Principe

Vitesse moyenne v_i en un point M_i entre deux positions proches :

$$v_i = \frac{M_{i-1}M_{i+1}}{t_{i+1} - t_{i-1}}$$

Le vecteur variation vitesse $\Delta\vec{v}_i$ au point M_i est défini par :

$$\Delta\vec{v}_i = \vec{v}_{i+1} - \vec{v}_{i-1}$$

Pour un système de masse m soumis à une ou plusieurs forces :

$$\boxed{\Sigma\vec{F} = m \times \frac{\Delta\vec{v}}{\Delta t}} \qquad \text{avec} \qquad \Delta = t_{i+1} - t_{i-1}$$

5.2.1.2 Exemple : lancé d'un ballon

```
import matplotlib.pyplot as plt
from math import sqrt

# MESURES
```

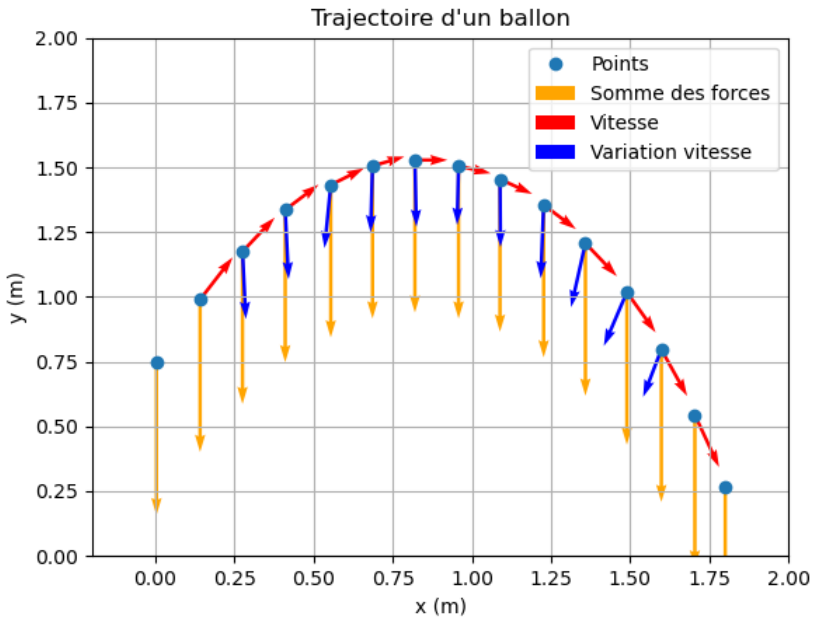
(suite sur la page suivante)

(suite de la page précédente)

```
t = [0.000, 0.066, 0.132, 0.198, 0.264, 0.330, 0.396, 0.462, 0.528, 0.594, 0.660, 0.726,
 ↪ 0.792, 0.858, 0.924]
x = [0.003, 0.141, 0.275, 0.410, 0.554, 0.686, 0.820, 0.958, 1.089, 1.227, 1.359, 1.490,
 ↪ 1.599, 1.705, 1.801]
y = [0.746, 0.990, 1.175, 1.336, 1.432, 1.505, 1.528, 1.505, 1.454, 1.355, 1.207, 1.018,
 ↪ 0.797, 0.544, 0.266]
N = len(x)

# CALCUL DES VITESSES
vx = [0 for i in range(N)]
vy = [0 for i in range(N)]

for i in range(1, N-1):
   vx[i] = (x[i+1]-x[i-1])/(t[i+1]-t[i-1])
   vy[i] = (y[i+1]-y[i-1])/(t[i+1]-t[i-1])

# CALCUL DES VARIATIONS VITESSE
dvx = [0 for i in range(N)]
dvy = [0 for i in range(N)]

for i in range(2, N-2):
   dvx[i] = vx[i+1]-vx[i-1]
   dvy[i] = vy[i+1]-vy[i-1]

dv = [round(sqrt(dvx[i]**2+dvy[i]**2),2) for i in range(N)]
print("Dv = ", dv)

# CALCUL DES SOMMES DES FORCES (poids uniquement)
m = 0.3
g = 9.81
Fx = [0 for i in range(N)]
Fy = [-m*g for i in range(N)]

# FIGURES
plt.plot(x, y, "o", label="Points")
plt.quiver(x, y, Fx,  Fy,  angles='xy', scale_units='xy', scale=5,  color='orange',␣
 ↪width=0.005, label="Somme des forces")
plt.quiver(x, y, vx,  vy,  angles='xy', scale_units='xy', scale=20, color='red',   ␣
 ↪width=0.005, label="Vitesse")
plt.quiver(x, y, dvx, dvy, angles='xy', scale_units='xy', scale=5,  color='blue',  ␣
 ↪width=0.005, label="Variation vitesse")
plt.legend()
plt.title("Trajectoire d'un ballon")
plt.xlabel("x (m)")
plt.xlim(-0.2, 2)
plt.ylabel("y (m)")
plt.ylim(0, 2)
plt.grid()
plt.show()
```

— La fonction `quiver()` permet de tracer un vecteur ou un champ de vecteur à partir de listes.

Résultats



5.2.2 Évolution d'un système chimique

Programme de première générale - Enseignement de spécialité - 2019

"Déterminer la composition de l'état final d'un système siège d'une transformation chimique totale à l'aide d'un langage de programmation.

5.2.2.1 Exemple 1 : quantités de matière à l'état final

On considère la réaction totale entre l'hydrogène sulfureux (H2S) et le dioxyde de soufre (SO2) qui produit du soufre et de l'eau et modélisée par l'équation :

$$2H_2S + SO_2 -> 3S + 2H_2O$$

```
a, b, c, d = 2, 1, 3, 2                    # coefficient stoechiométrique de H2S
n_H2S, n_SO2, n_S, n_H2O = 1, 1, 0, 0      # Quantités de matière initiales

x = 0           # Initialisation de l'avancement
dx = 0.01       # Pas de l'avancement

while (n_H2S>0) and (n_SO2>0) :
    x = x + dx                  # Incrémentation de l'avancement avec le pas
    n_H2S = n_H2S - a*dx        # Décrémentation des quantités de matière des réactifs
    n_SO2 = n_SO2 - b*dx        # ---
    n_S   = n_S   + c*dx        # Incrémentation des quantités de matière des réactifs
    n_H2O = n_H2O + d*dx        # ---

print('Avancement final = ',round(x,2), ' mol')
print('n(H2S) = ', round(n_H2S,2))
print('n(SO2) = ', round(n_SO2,2))
```

(suite sur la page suivante)

(suite de la page précédente)

```
print('n(S) = ',   round(n_S,2))
print('n(H2O) = ', round(n_H2O,2))
```

Résultats

```
Avancement final =  0.5  mol
n(H2S) =  -0.0
n(SO2) =  0.5
n(S) =  1.5
n(H2O) =  1.0
```

Questionnement

— Sous quelle(s) condition(s) la boucle `while` s'arrête-t-elle ?
— Quelle est l'influence de la grandeur `dx` ?
— Modifier les quantités de matière initiales pour obtenir un mélange stœchiométrique. Vérifier.

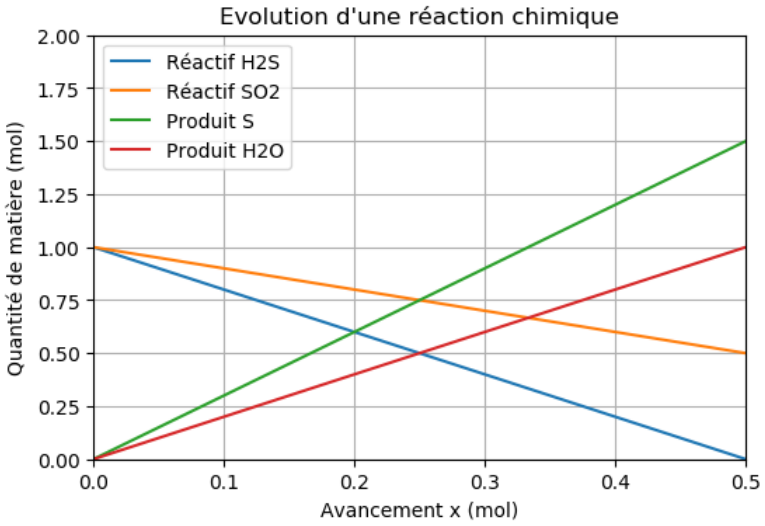
5.2.2.2 Exemple 2 : évolution des quantités de matière

Toujours pour la même réaction.

Pour un affichage sous forme de courbes, il faut mémoriser les quantités de matière dans des tableaux (liste).

```
import matplotlib.pyplot as plt

a, b, c, d = 2, 1, 3, 2                                          # coefficient␣
 ↪stoechiométrique de H2S
n_H2S, n_SO2, n_S, n_H2O = 1, 1, 0, 0                            # Quantités de␣
 ↪matière initiales
n_H2S, n_SO2, n_S, n_H2O = [n0_H2S], [n0_SO2], [n0_S], [n0_H2O]  # Initialisation des␣
 ↪tableaux
x = [0]
dx = 0.01

while (n_H2S[-1]>0) and (n_SO2[-1]>0) :
    x.append(x[-1] + dx)
    n_H2S.append(n_H2S[-1] - a*dx)
    n_SO2.append(n_SO2[-1] - b*dx)
    n_S.append(n_S[-1]   + c*dx)
    n_H2O.append(n_H2O[-1] + d*dx)

xMax = x[-1]

plt.plot(x,n_H2S,label = "Réactif H2S")
plt.plot(x,n_SO2,label = "Réactif SO2")
plt.plot(x,n_S,label = "Produit S")
plt.plot(x,n_H2O,label = "Produit H2O")
plt.title("Evolution d'une réaction chimique")
plt.xlabel("Avancement x (mol)")
plt.xlim(0,xMax)
plt.ylabel("Quantité de matière (mol)")
plt.ylim(0,2)
plt.legend()
plt.grid()
plt.show()
```

résultats



5.2.3 Bilan énergétique d'un mouvement

Programme de première générale - Enseignement de spécialité - 2019

"Utiliser un langage de programmation pour effectuer le bilan énergétique d'un système en mouvement".

5.2.3.1 Principe

Calculer à partir de données expérimentales ou de façon théorique l'énergie cinétique et l'énergie potentielle de pesanteur d'un système en mouvement.

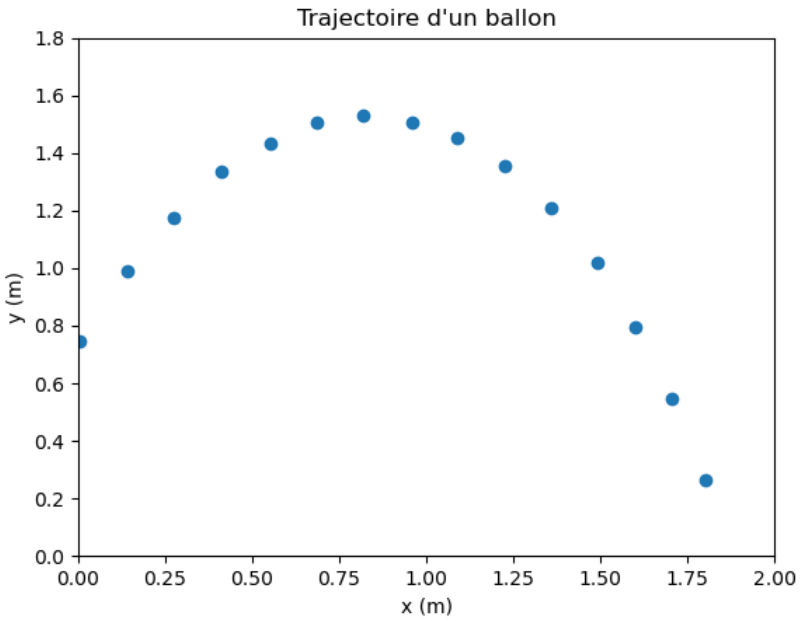
$$E_c = \frac{1}{2} m\ v^2 \qquad\qquad E_{pp} = m\ g\ h$$

Mettre en évidence la **conservation** ou la **non-conservation de l'énergie mécanique**.

$$E_m = E_c + E_{pp}$$

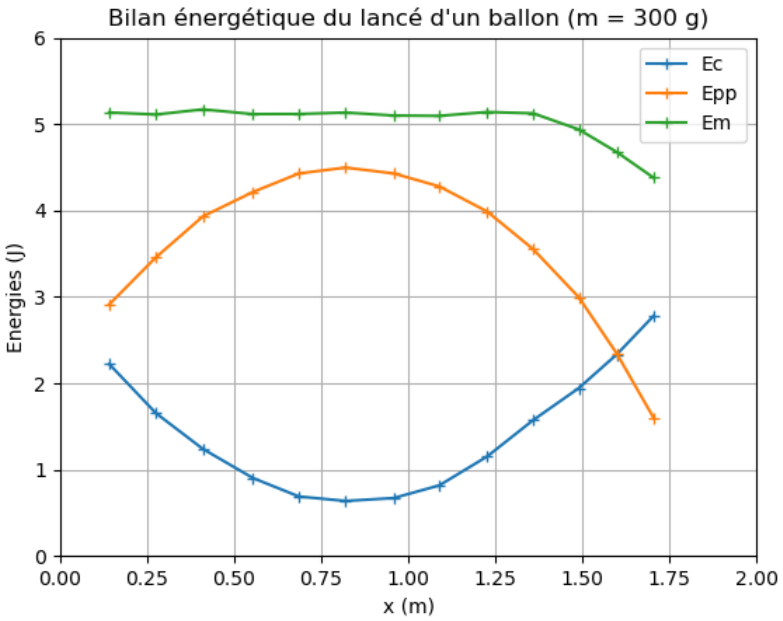
5.2.3.2 Exemple 1 : lancé d'un ballon

La trajectoire expérimentale d'un ballon est donnée par la figure suivante :




```
import matplotlib.pyplot as plt

# DONNEES EXPERIMENTALES (v a été calculée à partir des positions x et y)
x = [0.141, 0.275, 0.410, 0.554, 0.686, 0.820, 0.958, 1.089, 1.227, 1.359, 1.490, 1.599,
 ↪ 1.705]
t = [0.066, 0.132, 0.198, 0.264, 0.330, 0.396, 0.462, 0.528, 0.594, 0.660, 0.726, 0.792,
 ↪ 0.858]
v = [3.848, 3.320, 2.874, 2.452, 2.142, 2.061, 2.114, 2.333, 2.772, 3.238, 3.599, 3.943,
 ↪ 4.304]
z = [0.990, 1.175, 1.336, 1.432, 1.505, 1.528, 1.505, 1.454, 1.355, 1.207, 1.018, 0.797,
 ↪ 0.544]

# PARAMETRES
m = 0.3  # (kg)
g = 9.81 # (m/s²)

# CALCULS
N = len(v)                                  # Taille des tableaux
Ec  = [0.5*m*v[i]**2 for i in range(N)]     # Calcul de Ec
Epp = [m*g*z[i] for i in range(N)]          # Calcul de Epp
Em  = [Ec[i] + Epp[i] for i in range(N)] # Calcul de Em

# TRACES DES COURBES
plt.plot(x, Ec,  "+-", label="Ec")
plt.plot(x, Epp, "+-", label="Epp")
plt.plot(x, Em,  "+-", label="Em")
plt.legend()
plt.title("Bilan énergétique du lancé d'un ballon (m = 300 g)")
plt.xlabel("x (m)")
plt.xlim(0, 2)
plt.ylabel("Energies (J)")
plt.ylim(0, 6)
plt.grid()
plt.show()
```

Résultats



5.2.3.3 Exemple 2 : saut à l'élastique

D'après un document Eduscol.

Début de la chute (élastique non tendu) La hauteur de chute h (grandeur positive) est la distance entre l'objet de masse m en chute libre et le pont. La hauteur entre le pont et le sol est $h_0 = 80$ m.

L'altitude z du pont est prise comme référence d'où :

$$E_{pp} = m\ g\ z = -m\ g\ h$$

Par conservation de l'énergie mécanique on en déduit :

$$E_c = E_m - E_{pp} \qquad \text{avec} \qquad E_m = 0 \text{ J} \quad \text{(conditions initiales)}$$

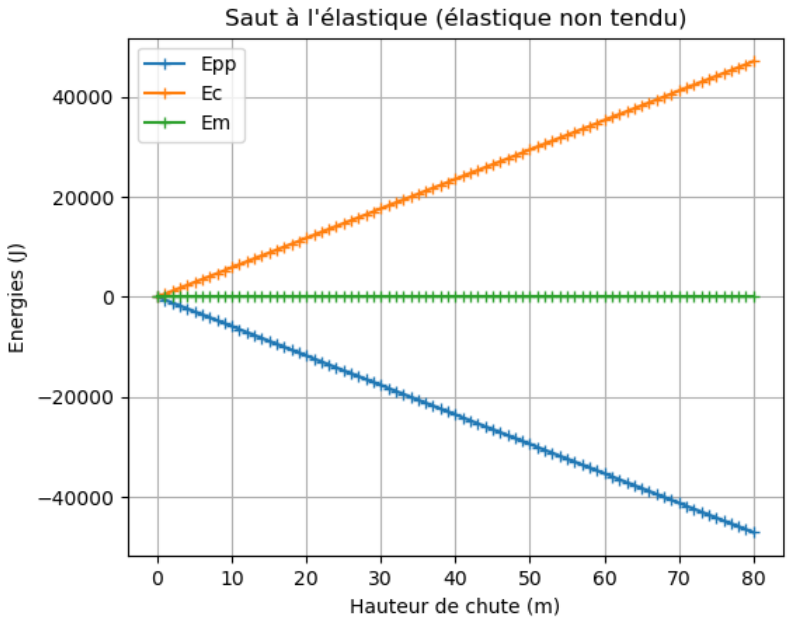
```
import matplotlib.pyplot as plt

# PARAMETRES
m = 60              # (kg)   Masse
g = 9.81            # (m/s²) Accélération de la pesanteur
h0 = 80             # (m)    Hauteur de la chute entre le pont et le sol

# INITIALISATION
h_chute = []        # (m) Tableau vide des hauteurs de chute
E_pp = []           # (J) Tableau vide des énergies potentielles de pesanteur
E_c = []            # (J) Tableau vide des énergies cinéitiques



# CALCULS DES ENERGIES
Em = 0              # (J) Energie mécanique aux conditions initiales
h = 0               # (m) Hauteur de chute initiale
pas = 1             # (m) Pas d'incrémentation de la hauter de chute
```

(suite sur la page suivante)

(suite de la page précédente)

```
while h <= h0:
    Epp = -m*g*h            # Calcul Epp
    Ec = Em - Epp           # Calcul Ec par conservation de l'énergie mécanique
    E_pp.append(Epp)        # Complète le tableau E_pp
    E_c.append(Ec)          # Complète le tableau E_c
    h_chute.append(h)       # Complète le tableau des hauteurs de chute
    h = h + pas             # Incrémente la hauteur

E_m = [E_c[i]+E_pp[i] for i in range(len(E_c))]  # Vérification de l'énergie mécanique

# COURBES
plt.subplots_adjust(left=0.17)              # Marge de 17% à gauche pour meilleur␣
 ↪affichage
plt.plot(h_chute, E_pp, "+-", label="Epp")  # Courbe de Epp
plt.plot(h_chute, E_c,  "+-", label="Ec")   # Courbe de Ec
plt.plot(h_chute, E_m,  "+-", label="Em")   # Courbe de Em
plt.title("Saut à l'élastique (élastique non tendu)")
plt.xlabel("Hauteur de chute (m)")
plt.ylabel("Energies (J)")
plt.legend()
plt.grid()
plt.show()
```

Résultats



Chute complète (élastique tendu) En tenant compte de l'action de l'élastique pour une distance supérieure à sa longueur à vide $l_0 = 24$ m.

L'énergie potentielle élastique s'écrit :

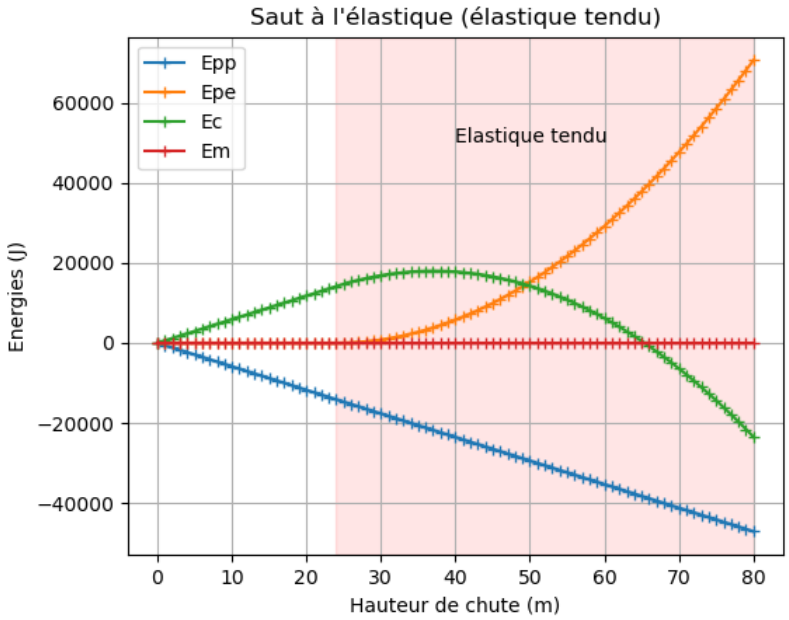
$$E_{pe} = \frac{1}{2}\, k\, (h - l_0)^2$$

Par conservation de l'énergie mécanique on en déduit :

$$E_c = E_m - E_{pp} - E_{pe} \qquad \text{avec} \qquad E_m = 0 \text{ J} \quad \text{(conditions initiales)}$$

```
import matplotlib.pyplot as plt

# PARAMETRES
m = 60              # (kg)   Masse
g = 9.81            # (m/s²) Accélération de la pesanteur
h0 = 80             # (m)    Hauteur de la chute entre le pont et le sol
l0 = 24             # (m)     Longueur de l'élastique
k = 45              # (N.m-1) Constante d'élasticité

# INITIALISATION
h_chute = []        # (m) Tableau vide des hauteurs de chute
E_pp = []           # (J) Tableau vide des énergies potentielles de pesanteur
E_pe = []           # (J) Tableau vide des énergies potentielles élastiques
E_c = []            # (J) Tableau vide des énergies cinéitiques



# CALCULS DES ENERGIES
Em = 0              # (J) Energie mécanique aux conditions initiales
h = 0               # (m) Hauteur de chute initiale
pas = 1             # (m) Pas d'incrémentation de la hauter de chute
while h <= h0:
   Epp = -m*g*h                # Calcul Epp
   if h<l0:
      Epe = 0                 # Epe (élastique non tendu)
   else:
      Epe = 0.5*k*(h-l0)**2  # Caclul Epe (élastique tendu)
   Ec = Em - Epp - Epe
   E_pp.append(Epp)            # Complète le tableau E_pp
   E_pe.append(Epe)            # Complète le tableau E_pe
   E_c.append(Ec)              # Complète le tableau E_c par conservation de l'énergie␣
 ↪mécanique
   h_chute.append(h)           # Complète le tableau des hauteurs de chute
   h = h + pas                 # Incrémente la hauteur

E_m = [E_c[i]+E_pp[i]+E_pe[i] for i in range(len(E_c))]  # Vérification de l'énergie␣
 ↪mécanique

# COURBES
plt.subplots_adjust(left=0.17)             # Marge de 17% à gauche pour meilleur␣
 ↪affichage
plt.plot(h_chute, E_pp, "+-", label="Epp")  # Courbe de Epp
plt.plot(h_chute, E_pe, "+-", label="Epe")  # Courbe de Epe
plt.plot(h_chute, E_c,  "+-", label="Ec")   # Courbe de Ec
plt.plot(h_chute, E_m,  "+-", label="Em")   # Courbe de Em
plt.axvspan(l0, h0, color='r', alpha=0.1) # Zone où l'élastique est tendu
plt.text(40, 5E4, "Elastique tendu")
plt.title("Saut à l'élastique (élastique tendu)")
plt.xlabel("Hauteur de chute (m)")
plt.ylabel("Energies (J)")
plt.legend()
plt.grid()
plt.show()
```

Résultats



Questionnement

— Quelle est la vitesse maximale de la chute ?
— Est-il possible physiquement d'obtenir une énergie cinétique négative ?
— Ce saut à l'élastique est-il sûr ? Quelle est la hauteur maximale de la chute ?
— Modifier le code pour obtenir la hauteur maximale.

Détermination de la hauteur maximale Il suffit d'ajouter une condition supplémentaire sur une énergie cinétique toujours positive ou nulle.

La hauteur maximale est la dernière valeur du tableau des hauteurs (avant dernière en réalité).

```
import matplotlib.pyplot as plt

# PARAMETRES
m = 60              # (kg)   Masse
g = 9.81            # (m/s²) Accélération de la pesanteur
h0 = 80             # (m)    Hauteur de la chute entre le pont et le sol
l0 = 24             # (m)     Longueur de l'élastique
k = 45              # (N.m-1) Constante d'élasticité

# INITIALISATION
h_chute = []        # (m) Tableau vide des hauteurs de chute
E_pp = []           # (J) Tableau vide des énergies potentielles de pesanteur
E_pe = []           # (J) Tableau vide des énergies potentielles élastiques
E_c = []            # (J) Tableau vide des énergies cinéitiques



# CALCULS DES ENERGIES
Em = 0              # (J) Energie mécanique aux conditions initiales
Ec = 0              # (J) Energie cinétique initiale
h = 0               # (m) Hauteur de chute initiale
pas = 0.1           # (m) Pas d'incrémentation de la hauter de chute
```

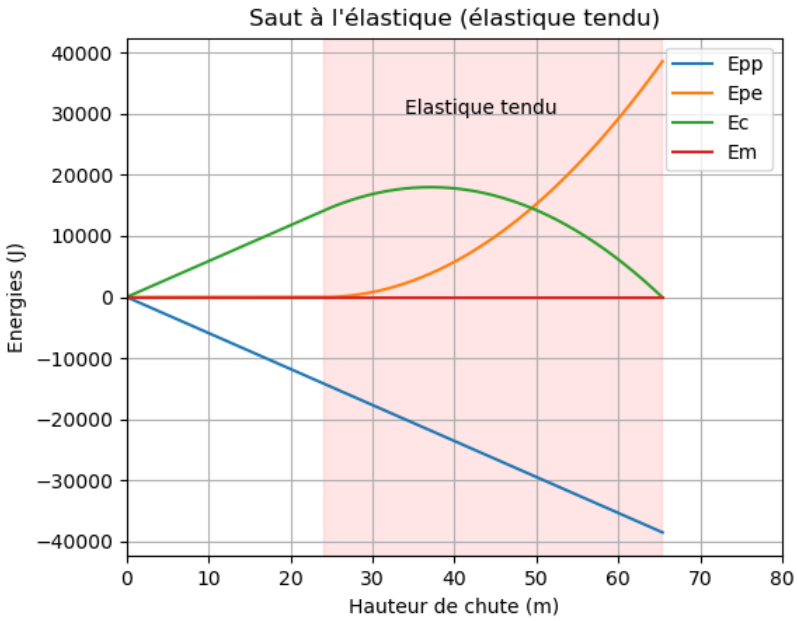
(suite sur la page suivante)

(suite de la page précédente)

```
while h<=h0 and Ec>=0:
   Epp = -m*g*h                # Calcul Epp
   if h<l0:
      Epe = 0                  # Epe (élastique non tendu)
   else:
      Epe = 0.5*k*(h-l0)**2    # Caclul Epe (élastique tendu)
   Ec = Em - Epp - Epe
   E_pp.append(Epp)            # Complète le tableau E_pp
   E_pe.append(Epe)            # Complète le tableau E_pe
   E_c.append(Ec)              # Complète le tableau E_c par conservation de l'énergie␣
 ↪mécanique
   h_chute.append(h)           # Complète le tableau des hauteurs de chute
   h = h + pas                 # Incrémente la hauteur

hmax = h_chute[-2]  # La hauteur maximale est avant dernière vauleur
print("La hauteur maximale de la chute est de", round(hmax,1), "m")

E_m = [E_c[i]+E_pp[i]+E_pe[i] for i in range(len(E_c))]  # Vérification de l'énergie␣
 ↪mécanique

# COURBES
plt.subplots_adjust(left=0.17)              # Marge de 17% à gauche pour meilleur␣
 ↪affichage
plt.plot(h_chute, E_pp, "-", label="Epp")  # Courbe de Epp
plt.plot(h_chute, E_pe, "-", label="Epe")  # Courbe de Epe
plt.plot(h_chute, E_c,  "-", label="Ec")   # Courbe de Ec
plt.plot(h_chute, E_m,  "-", label="Em")   # Courbe de Em
plt.axvspan(l0, hmax, color='r', alpha=0.1) # Zone où l'élastique est tendu
plt.text(l0+10, 3E4, "Elastique tendu")
plt.title("Saut à l'élastique (élastique tendu)")
plt.xlabel("Hauteur de chute (m)")
plt.ylabel("Energies (J)")
plt.xlim(0, h0)
plt.legend()
plt.grid()
plt.show()
```

Résultats

```
>>> %Run
La hauteur maximale de la chute est de 65.3 m
```



5.2.4 Représentation d'une onde

Programme de première générale - Enseignement de spécialité - 2019

"Représenter un signal périodique et illustrer l'influence de ses caractéristiques (période, amplitude) sur sa représentation".

5.2.4.1 Principe

Tracer un signal sinusoïdal décrit par l'expression :

$$s(t) = A \times \cos(\frac{2\pi}{T} \cdot t)$$

Montrer l'influence de la période et de l'amplitude.

5.2.4.2 Exemple 1 : pour une amplitude et une période

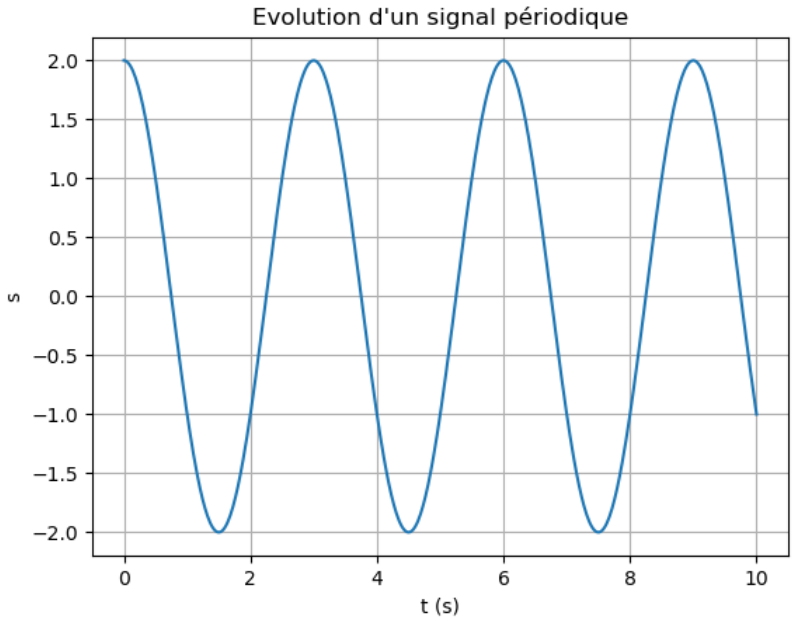
Une simple représentation d'un signal sinusoïdal. Les valeurs des variables `A` (amplitude) et `T` (période) sont à modifier pour montrer leurs influences.

```
import numpy as np
import matplotlib.pyplot as plt
from math import pi

# DEFINTION DES CONSTANTES
A = 2     # Amplitude
T = 3     # Période (s)

# CALCULS
t = np.linspace(0, 10, 500)  # Tableau des valeurs du temps en seconde
```

(suite sur la page suivante)

(suite de la page précédente)

```
s = A*np.cos(2*pi*t/T)       # Tableau des valeurs calculées du signal sinusoïdal

# COURBES
plt.plot(t, s)
plt.title("Evolution d'un signal périodique")
plt.xlabel("t (s)")
plt.ylabel("s")
plt.grid()
plt.savefig("premiere_signal_sinusoidal_1.png")
plt.show()
```



5.2.4.3 Exemple 2 : pour plusieurs amplitudes

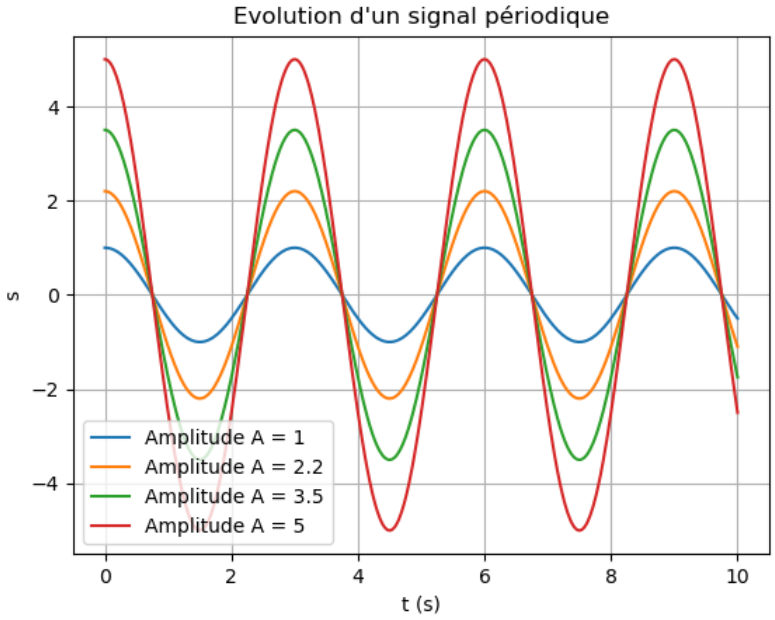
Tracer plusieurs signaux sinusoïdaux pour des amplitudes différentes placée dans une liste à modifier !

```
import numpy as np
import matplotlib.pyplot as plt
from math import pi

# DEFINTION DES CONSTANTES
T = 3     # Période (s)

# CALCULS
t = np.linspace(0, 10, 500)  # Tableau des valeurs du temps en seconde

# COURBES
for A in [1, 2.2, 3.5, 5]:
   s = A*np.cos(2*pi*t/T)       # Tableau des valeurs calculées du signal sinusoïdal
   plt.plot(t, s, label="Amplitude A = " + str(A))

plt.legend()
plt.title("Evolution d'un signal périodique")
plt.xlabel("t (s)")
```

(suite sur la page suivante)

(suite de la page précédente)

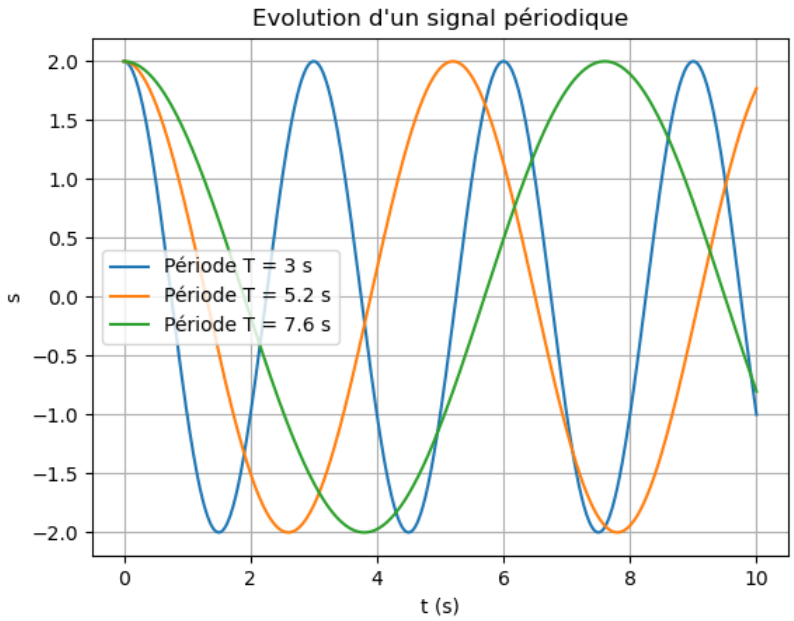
```
plt.ylabel("s")
plt.grid()
plt.show()
```



5.2.4.4 Exemple 3 : pour plusieurs périodes

Tracer plusieurs signaux sinusoïdaux pour des périodes différentes placée dans une liste à modifier !

```
import numpy as np
import matplotlib.pyplot as plt
from math import pi

# DEFINTION DES CONSTANTES
A = 2     # Amplitude

# CALCULS
t = np.linspace(0, 10, 500)  # Tableau des valeurs du temps en seconde

# COURBES
for T in [3, 5.2, 7.6]:
   s = A*np.cos(2*pi*t/T)       # Tableau des valeurs calculées du signal sinusoïdal
   plt.plot(t, s, label="Période T = " + str(T) + " s")

plt.legend()
plt.title("Evolution d'un signal périodique")
plt.xlabel("t (s)")
plt.ylabel("s")
plt.grid()
plt.show()
```



5.2.4.5 Exemple 4 : avec des curseurs

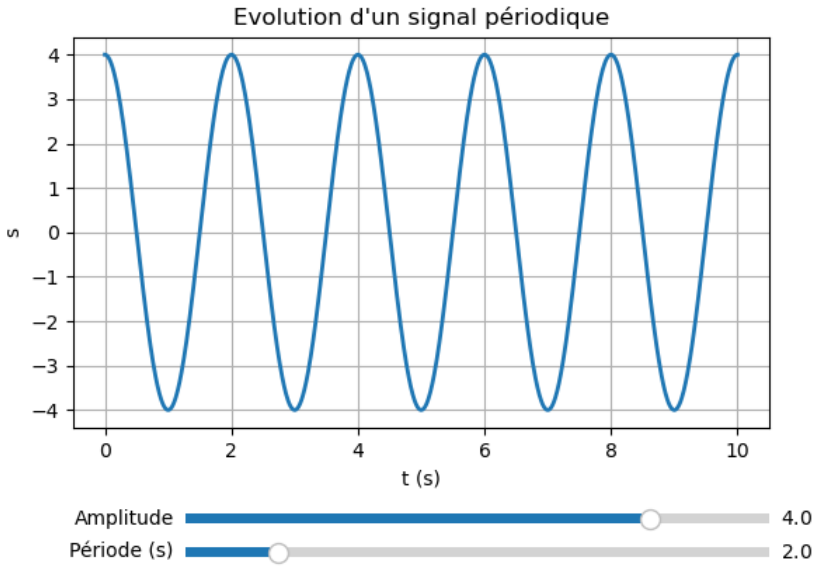
Modifier l'amplitude et/ou la période à l'aide de la souris avec des curseurs (sliders).

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider # Curseur
from math import pi

# DEFINITION DES CONSTANTES
A = 4     # Amplitude initiale
T = 2     # Période initiale
dT = 0.1  # Pas de variation de la période

# CALCULS
t = np.linspace(0, 10, 1000)  # Tableau des valeurs du temps en seconde
s = A*np.cos(2*pi*t/T)        # Tableau des valeurs calculées du signal sinusoïdal

# COURBE
fig, ax = plt.subplots()          # Zone de dessin
plt.subplots_adjust( bottom=0.30) # Laisse une marge de 30% en bas de la zone de dessin␣
 ↪pour placement des curseurs

trace, = plt.plot(t, s, lw=2)                 # Trace le signal s(t) et récupère le␣
 ↪tracé
plt.title("Evolution d'un signal périodique") # Titre
plt.xlabel("t (s)")                           # Légende axe x
plt.ylabel("s")                               # Légende axe y
plt.grid()                                    # Affiche la grille


# AJOUT DES CURSEURS
ax_periode = plt.axes([0.25, 0.1, 0.65, 0.03], facecolor='lightgoldenrodyellow') # Zone␣
 ↪de placement du curseur période
```

(suite sur la page suivante)

(suite de la page précédente)

```
ax_amp = plt.axes([0.25, 0.15, 0.65, 0.03], facecolor='lightgoldenrodyellow')     # Zone␣
 ↪de placement du curseur amplitude
s_periode = Slider(ax_periode, 'Période (s)', 0.5, 10.0, valinit=T, valstep=dT)  #␣
 ↪Paramétrage du curseur période
s_amp = Slider(ax_amp, 'Amplitude', 0.1, 5, valinit=A)                           #␣
 ↪Paramétrage du curseur amplitude

def update(val):
    """ Fonction d'actulisation de la courbe """
    A = s_amp.val        # Récupération de la nouvelle amplitude
    T = s_periode.val    # Récupération de la nouvelle période
    trace.set_ydata(A*np.cos(2*pi*t/T))   # Calcul du nouveau signal
    fig.canvas.draw_idle()                # Dessine la nouvelle courbe

s_periode.on_changed(update)    # Lance la fonction update(val) si le curseur période␣
 ↪change !
s_amp.on_changed(update)        # Lance la fonction update(val) si le curseur amplitude␣
 ↪change !

plt.show()
```



5.2.5 Simuler la propagation d'une onde

Programme de première générale - Enseignement de spécialité - 2019

"Simuler à l'aide d'un langage de programmation, la propagation d'une onde périodique".

5.2.5.1 Principe

Le but est de simuler la propagation d'une onde sinusoïdale dans le sens des x croissants à partir de sa **célérité** c :

$$s(x,t) = A \cdot \cos[\frac{2\pi}{T}(t - \frac{x}{c})]$$

ou de sa **longueur d'onde** λ :

$$s(x,t) = A \cdot \cos[2\pi(\frac{t}{T} - \frac{x}{\lambda})]$$

Y mettre en évidence :

— la période temporelle T ;
— la période spatiale λ ;
— la célérité $c = \frac{\lambda}{T}$.

5.2.5.2 Exemple : animation avec pause par un clic de la souris

Cet exemple propose une animation de la propagation d'un onde sinusoïdal à partir de sa période temporelle T et de sa célérité c.

Un **clic de la souris** sur la figure permet de **stopper ou de reprendre l'animation**.

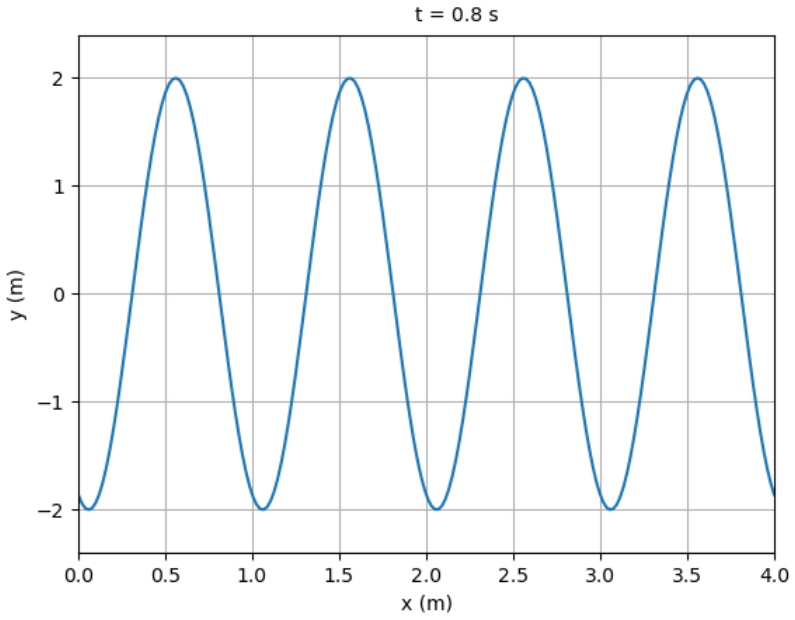
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from math import pi

# PARAMETRES DE L'ONDE A MODIFIER
T = 1      # (s)   Période
c = 1      # (m/s) Célérité
A = 2      # (m)   Amplitude

# PARAMETRES DU TRACÉ
nb_t = 1000  # Nombre de valeurs de temps
Dt = 0.01    # Pas d'incrémentation du temps

nb_x = 200   # Nombre de valeurs de x
x_max = 4    # Valeur maximale de x

# TABLEAU DE VALEURS DE x
x = np.linspace(0, x_max, nb_x)

# PAUSE/LECTURE DE L'ANIMATION (clic de la souris sur la courbe)
pause = True
def onClick(event):
   global pause
   if pause:
      ani.event_source.stop()
      pause = False
   else:
      ani.event_source.start()
      pause = True

# FONTION D'ANIMATION : calcul du signal sinusoïdal
def actualise_onde(i):
```

(suite sur la page suivante)

(suite de la page précédente)

```
    t = i * Dt                                          # Calcul de l'instant t en cours
    y = A * np.sin(2*np.pi/T*(t - x/c))                 # Calcul de l'onde en fonction de x␣
 ↪à l'instant t en cours
    courbe.set_data(x, y)                               # Actualise le tracé de la courbe
    text.set_text("t = " + str(round(t,1)) + " s")      # Actualise l'affichage du temps
    return courbe, text

# FIGURE
fig, ax = plt.subplots()                                  # Initialise une figure et␣
 ↪récupère le repère (ax)
fig.canvas.mpl_connect('button_press_event', onClick) # Activation de la gestion du␣
 ↪clic de la souris
text = fig.text(0.5,0.90, "")                             # Zone de text pour afficher le␣
 ↪temps

courbe, = plt.plot([],[])                                 # Initialise un tracé vide et␣
 ↪récupère la référence de la courbe.

plt.grid()              # Affiche la grille
plt.xlim(0, x_max)      # Echelle sur l'axe des x
plt.xlabel("x (m)")     # Légende sur l'axe des x
plt.ylim(-1.2*A, 1.2*A) # Echelle sur l'axe des y
plt.ylabel("y (m)")     # Légende sur l'axe des y

ani = animation.FuncAnimation(fig, actualise_onde, frames=nb_t, interval=10,␣
 ↪repeat=False) # Active l'animation

plt.show()  # Affiche la courbe
```

Résultats



Questionnement

— Déterminer la longueur d'onde λ.
— Est-elle en accord avec les paramètres du programme ?
— Modifier le programme pour une animation à partir de la fréquence f plutôt que la période T.

5.3 Classe terminale, enseignement de spécialité

5.3.1 Histogramme d'une série de mesure

Programme de classe terminale, enseignement de spécialité, voie générale

Représenter l'histogramme associé à une série de mesures à l'aide d'un tableur ou d'un langage de programmation.

5.3.1.1 Principe

Tracer l'histogramme d'une série de n mesures d'une grandeur x.

Déterminer la **valeur moyenne** :

$$\bar{x} = \frac{x_1 + x_2 + ... + x_n}{n}$$

l'écart-type :

$$s = \sqrt{\frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + ... + (x_n - \bar{x})^2}{n-1}}$$

et **l'incertitude-type** :

$$u(\bar{x}) = \frac{s}{\sqrt{n}}$$

En langage Python, plusieurs librairies proposent des fonctions statistiques.

Module	Fonction moyenne	Fonction écart-type
`numpy`	`mean(x)`	`std(x, ddof=1)`
`statistics`	`mean(x)`	`stdev(x)`

5.3.1.2 Exemple : célérité d'un son

Soit une série de mesure de la célérité du son donnée par un module ultrason associé à un Arduino.

```
import matplotlib.pyplot as plt
from statistics import mean, stdev, sqrt

# MESURES
c= [352.18, 349.31, 350.47, 349.34, 350.43, 350.47, 349.52, 349.52, 348.57,
   349.52, 349.31, 349.52, 349.52, 349.34, 349.55, 347.70, 349.52, 349.48,
   349.31, 348.60, 350.47, 349.52, 348.60, 349.31, 348.57, 349.52, 349.31,
   349.48, 348.53, 348.60]
n = len(c)

# CALCULS
c_moy = mean(c)     # valeur moyenne
s = stdev(c)        # écart-type standard
u = s/sqrt(n)       # incertitude-type

# AFFICHAGE
print("c_moy =", c_moy, "s =", s, "u(c) =", u)
print("c =", round(c_moy,1), "+/-", round(u,1), "m/S")
```

(suite sur la page suivante)

(suite de la page précédente)

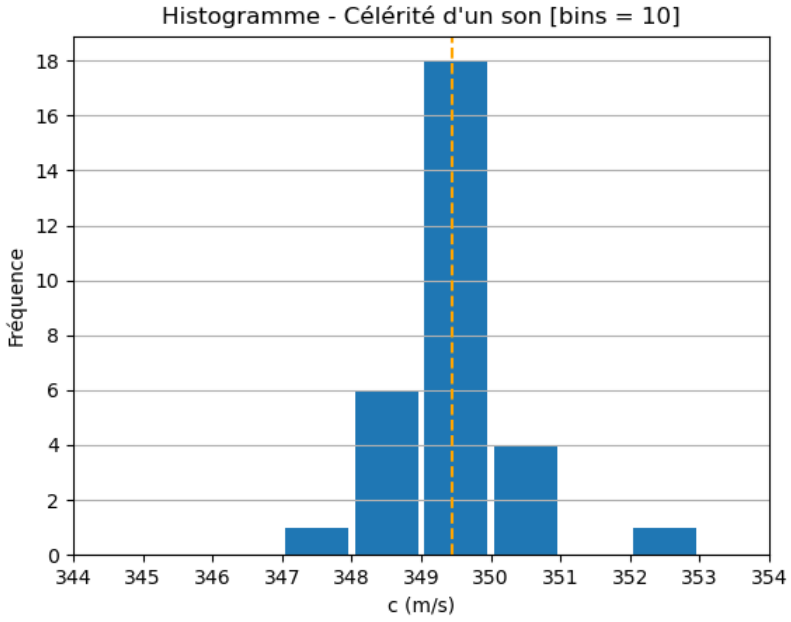
```
# HISTOGRAMME
plt.hist(c, range=(344,354), bins=10, rwidth=0.9)          # Trace l'histogramme
plt.axvline(c_moy, color='orange', ls='--')                # Ligne verticale pour la␣
 ↪valeur moyenne
plt.title("Histogramme - Célérité d'un son [bins = 10]") # Titre
plt.ylabel('Fréquence')                                    # Légende en y
plt.xlabel("c (m/s)")                                      # Légende en x
plt.xlim(344, 354)                                         # Echelle en x
plt.xticks([i for i in range(344, 355)])                   # Graduation manuelle en x
plt.yticks([i for i in range(0, 20, 2)])                   # Graduation manuelle en y
plt.grid(axis='y')                                         # Grille uniquement en y
plt.show()                                                 # Affiche la figure
```

— `range=(344, 354)` fixe les limites de la plage d'étude du tableau de données.
— `bins=10` est le nombre d'intervalles dans la plage d'étude.

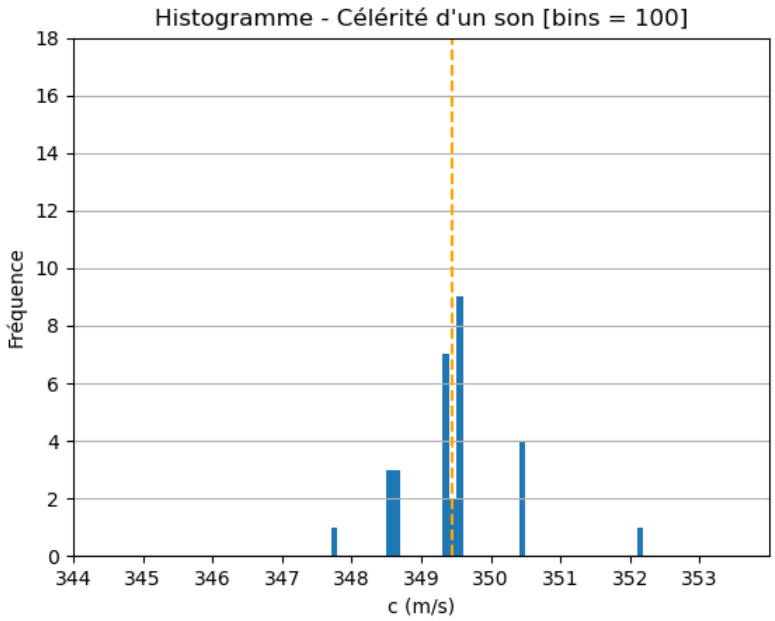
Résultats

```
>>> %Run script.py
c_moy = 349.4363333333333 s = 0.8163690989663521 u(c) = 0.14904792358467947
c = 349.4 +/- 0.1 m/s
```

Pour l'option `bins = 10` :



Pour l'option `bins = 100` :



5.3.2 Incertitudes-types composées - Simulation

Programme de classe terminale, enseignement de spécialité, voie générale

Simuler, à l'aide d'un langage de programmation, un processus aléatoire illustrant la détermination de la valeur d'une grandeur avec incertitudes-types composées.

5.3.2.1 Principe

Estimer une incertitude-type à partir de plusieurs sources d'erreurs à l'aide d'une simulation Python utilisant un **processus aléatoire** (module `random`) sur un **grand nombre d'expériences**.

Comparer la valeur obtenue à l'incertitude-type donnée par la **formule de composition des incertitudes**.

5.3.2.2 Exemple : titrage

Soit le titrage d'un acide de concentration C_A et de volume V_A par une base de concentration C_B et de volume V_B.

A l'équivalence :

$$V_B = V_E \qquad \text{et} \qquad \boxed{C_A = \frac{C_B \cdot V_E}{V_A}}$$

Données

Grandeur	Valeur	Source d'erreurs	Incertitude-type
C_B	0,70 mol/L	Préparation solution titrante	0,02 mol/L
V_A	10,00 mL	Pipette jaugée 10,0 mL (classe A)	0,02 mL
V_E	20,00 mL	Burette graduée de 25 mL	0,03 mL

Le programme suivant **simule la variabilité de le concentration C_A pour un nombre important de titrages** en tenant compte de l'incertitude-type de chaque grandeur présente dans l'expression de C_A.

```
import numpy as np
from statistics import mean, stdev

def alea(x):
   """ Retourne une valeur avec une erreur aléatoire de la grandeur x
   """
   tirage = np.random.normal() # Tirage suivant une loi normale entre - infini et +␣
↪infini
   return x[0]+x[1]*tirage     # Valeur avec erreur aléatoire

CB = [0.70, 0.02]   # valeur et incertitude-type de CB
VA = [10.00, 0.02]  # valeur et incertitude-type de VA
VE = [20.00, 0.03]  # valeur et incertitude-type DE VE

n = 100             # Nombre de titrage : 100, 1000, 10000, ...
CA = [alea(CB)*alea(VE)/alea(VA) for i in range(n)] # Calcul de CA pour les n titrages

moy = mean(CA)      # Valeur moyenne
s = stdev(CA)       # Ecart-type

print("moy = ", moy, "s =", s)
print("Moyenne CA =", round(moy,2), "mol/L")
print("Incertitude type sur CA =", round(s,2), "mol/L")
```

Résultats

```
>>> %Run script.py
moy =  1.4023357675343335 s = 0.03913323674046289
Moyenne CA = 1.4 mol/L
Incertitude type sur CA = 0.04 mol/L

>>> %Run script.py
moy =  1.3920096197242249 s = 0.03937987403174825
Moyenne CA = 1.39 mol/L
Incertitude type sur CA = 0.04 mol/L
```

Plus grand sera `n`, meilleure sera l'estimation de l'incertitude-type composée !

Comparaison

$$u(C_A) = C_A \times \sqrt{(\frac{u(C_B)}{C_B})^2 + (\frac{u(V_E)}{V_E})^2 + (\frac{u(V_A)}{V_A})^2}$$

$$u(C_A) = 1,4 \times \sqrt{(\frac{0,02}{0,70})^2 + (\frac{0,03}{20,00})^2 + (\frac{0,02}{10,00})^2} \approx 0,04 \text{ mol/L}$$

5.3.3 Titrage - Evolution des quantités de matière

Programme de classe terminale, enseignement de spécialité, voie générale

Représenter, à l'aide d'un langage de programmation, l'évolution des quantités de matière des espèces en fonction du volume de solution titrante versé.

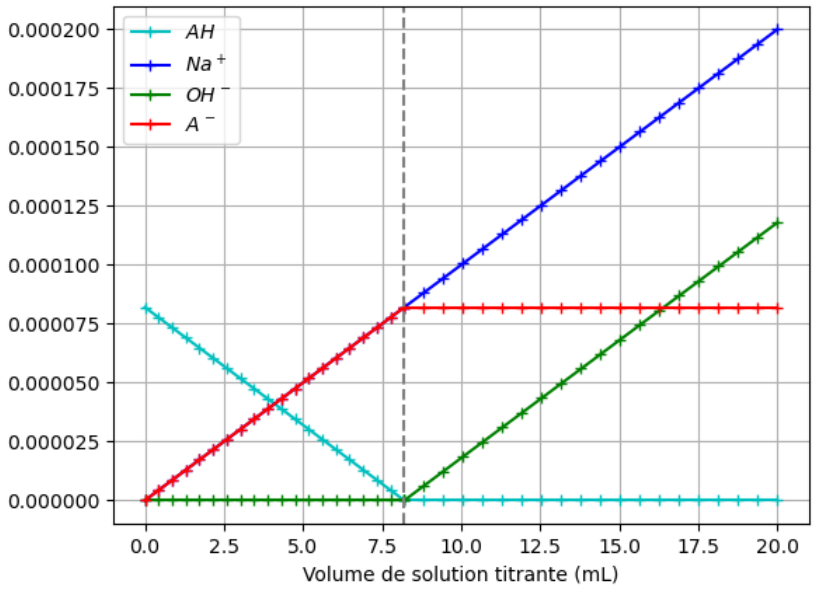
5.3.3.1 Dosage par titrage par suivi conductimétrie

Titrage par conductimétrie de l'acide ascorbique ($C_6H_8O_6$) par une solution d'hydroxyde de sodium / soude ($Na^+ + OH^-$).

Méthode 1 : tableaux Numpy sans conditionnelle `if` Évolution de quantités de matière des espèces en fonction du volume de solution titrante versé connaissant V_E et donc C_A.

```
"""
Titrage par conductimétrie de l'acide ascorbique (C6H8O6)
par une solution d'hydroxyde de sodium / soude (Na+ HO-)
"""
import numpy as np
import matplotlib.pyplot as plt

# VARIABLES
VA = 40.0     # (mL)    Volume de solution titrée
CB = 0.010    # (mol/L) Concentration de soude

VE = 8.2      # (mL)    Volume de soude versé à l'équivalence
CA = CB*VE/VA # (mol/L) Concentration initiale d'acide ascorbique

# AVANT EQUIVALENCE
VB1 = np.linspace(0, VE, 20)  # Valeurs de VB avant équivalence
nAH_1 = CA*VA*1E-3 - CB*VB1*1E-3
nNa_1 = CB*VB1*1E-3
nOH_1 = 0*VB1
nA_1 = CB*VB1*1E-3

# APRES EQUIVALENCE
VB2 = np.linspace(VE, 20, 20)  # Valeurs de VB avant équivalence
nAH_2 = 0*VB2
nNa_2 = CB*VB2*1E-3
nOH_2 = CB*(VB2-VE)*1E-3
nA_2 = 0*VB2 + CB*VE*1E-3

# COURBES
plt.plot(VB1, nAH_1, "c+-", label=r"$AH$")
plt.plot(VB2, nAH_2, "c+-")

plt.plot(VB1, nNa_1, "b+-", label=r"$Na^+$")
plt.plot(VB2, nNa_2, "b+-")

plt.plot(VB1, nOH_1, "g+-", label=r"$OH^-$")
plt.plot(VB2, nOH_2, "g+-")

plt.plot(VB1, nA_1, "r+-", label=r"$A^-$" )
plt.plot(VB2, nA_2, "r+-")

plt.axvline(VE, color="gray", linestyle ="dashed")

plt.xlabel("Volume de solution titrante (mL)")
plt.ylabel("Quantité de matière (mol)")
plt.legend()
plt.grid()
plt.show()
```

Résultats



Méthode 2 : listes avec conditionnelle `if` Évolution de quantités de matière des espèces en fonction du volume de solution titrante versé connaissant V_E et donc C_A.

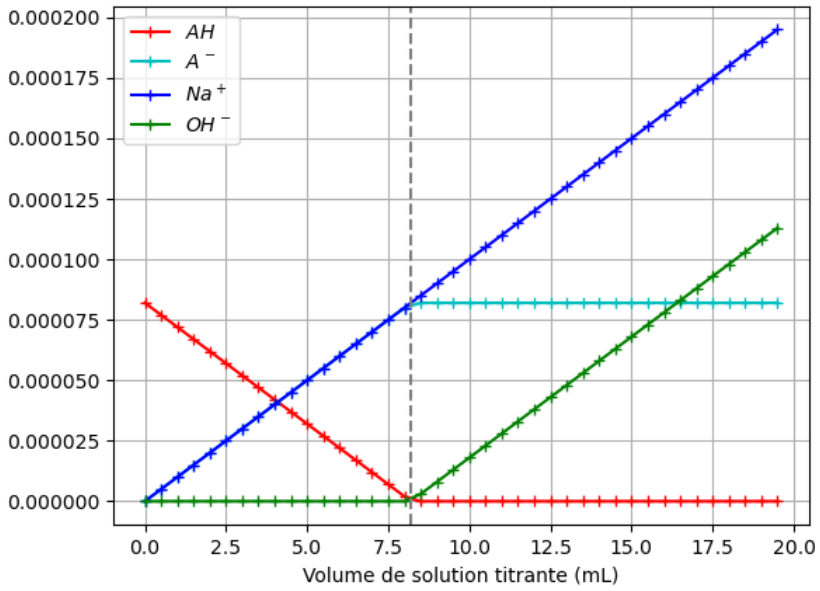
```
"""
Titrage par conductimétrie de l'acide ascorbique (C6H8O6)
par une solution d'hydroxyde de sodium / soude (Na+ HO-)
"""
import numpy as np
import matplotlib.pyplot as plt

# VARIABLES
VA = 40.0     # (mL)    Volume de solution titrée
CB = 0.010    # (mol/L) Concentration de soude

VE = 8.2      # (mL)    Volume de soude versé à l'équivalence
CA = CB*VE/VA # (mol/L) Concentration initiale d'acide ascorbique

VB = []
nAH = []
nNa = []
nOH = []
nA = []

Vmax = 20
N = 40

for i in range(0, N):
    VB.append(Vmax*i/N)
    if VB[i]<VE:
        nAH.append(CA*VA*1E-3 - CB*VB[i]*1E-3)
        nNa.append(CB*VB[i]*1E-3)
        nOH.append(0)
        nA.append(CB*VB[i]*1E-3)
    else:
        nAH.append(0)
```

(suite sur la page suivante)

(suite de la page précédente)

```
        nNa.append(CB*VB[i]*1E-3)
        nOH.append(CB*(VB[i]-VE)*1E-3)
        nA.append(CB*VE*1E-3)


# COURBES
plt.plot(VB, nAH, "r+-", label=r"$AH$")
plt.plot(VB, nA, "c+-", label=r"$A^-$" )
plt.plot(VB, nNa, "b+-", label=r"$Na^+$")
plt.plot(VB, nOH, "g+-", label=r"$OH^-$")

plt.axvline(VE, color="gray", linestyle ="dashed")

plt.xlabel("Volume de solution titrante (mL)")
plt.ylabel("Quantité de matière (mol)")
plt.legend()
plt.grid()
plt.show()
```

Résultats



Méthode 3 : pas à pas Évolution de quantités de matière des espèces en fonction du volume de solution titrante versé connaissant C_A.

Détermination de V_E.

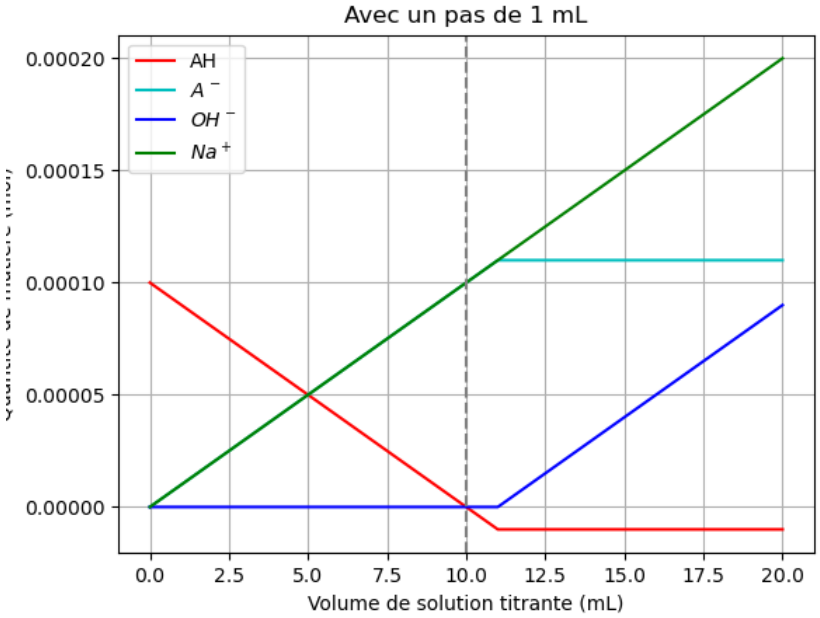
```
"""
Titrage par conductimétrie de l'acide ascorbique (C6H8O6)
par une solution d'hydroxyde de sodium ou soude (Na+ HO-)
Détermination de Ve connaissant CA
"""
import numpy as np
import matplotlib.pyplot as plt
```

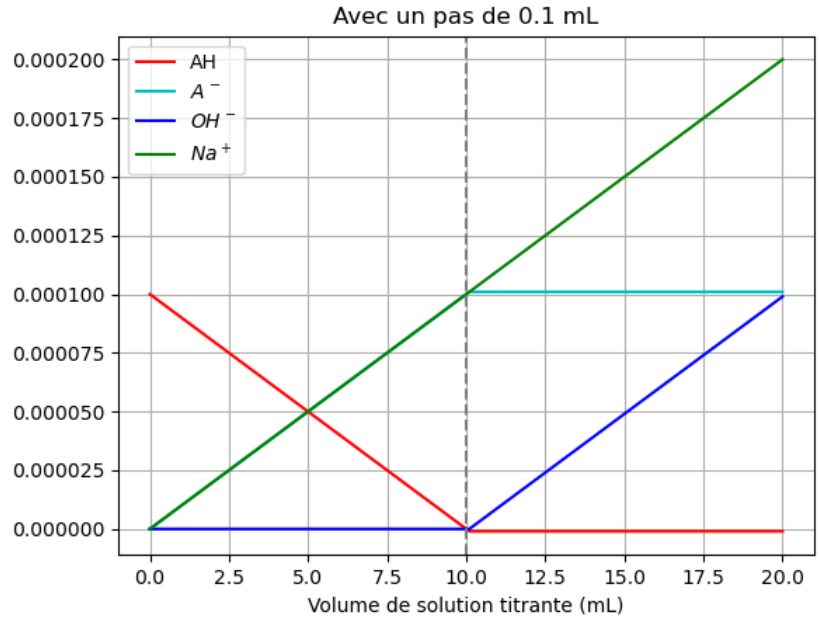
(suite sur la page suivante)

(suite de la page précédente)

```
# VARIABLES
CA = 0.0025
VA = 40.0     # (mL)    Volume de solution titrée
CB = 0.010    # (mol/L) Concentration de soude

VB = [0]
nAH = [CA*VA*1E-3]
nNa_plus = [0]
nOH_moins = [0]
nA_moins = [0]

VB_pas = 0.1 # mL

while VB[-1]<20:
   if nAH[-1]>0:
      nAH.append(nAH[-1] - CB*VB_pas*1E-3)           # Consommation AH
      nNa_plus.append(nNa_plus[-1] + CB*VB_pas*1E-3) # Apport de Na+
      nOH_moins.append(0)                            # OH- est totalement consommé
      nA_moins.append(nA_moins[-1] + CB*VB_pas*1E-3) #
      VE = VB[-1]
   else:
      nAH.append(nAH[-1])
      nNa_plus.append(nNa_plus[-1] + CB*VB_pas*1E-3)
      nOH_moins.append(nOH_moins[-1] + CB*VB_pas*1E-3)
      nA_moins.append(nA_moins[-1])
   VB.append(VB[-1]+VB_pas)

print("VE = ", VE, "mL")

# COURBES
plt.plot(VB, nAH, "r-", label="AH")
plt.plot(VB, nA_moins, "c-", label=r"$A^-$" )
plt.plot(VB, nOH_moins, "b-", label=r"$OH^-$")
plt.plot(VB, nNa_plus, "g-", label=r"$Na^+$")

plt.axvline(VE, color="gray", linestyle ="dashed")

plt.title("Avec un pas de " + str(VB_pas) + " mL")
plt.xlabel("Volume de solution titrante (mL)")
plt.ylabel("Quantité de matière (mol)")
plt.legend()
plt.grid()
plt.show()
```

Résultats





5.3.4 Evolution temporelle d'une transformation chimique

Programme de classe terminale, enseignement de spécialité, voie générale

À l'aide d'un langage de programmation et à partir de données expérimentales, tracer l'évolution temporelle d'une concentration, d'une vitesse volumique d'apparition ou de disparition et tester une relation donnée entre la vitesse volumique de disparition et la concentration d'un réactif.

5.3.4.1 Simulation de la disparition d'un espèce chimique

Principe Dans une réaction chimique d'ordre 1, la vitesse volumique de disparition d'un réactif R est proportionnelle à sa concentration [R] au cours du temps.

$$v_{disp}(R) = k \times [R] \quad \text{avec} \quad v_{disp}(R) = -\frac{d[R]}{dt}$$

k est la constante de vitesse.

Le calcul de la vitesse est donnée par l'expression :

$$v_{disp}(R) = \frac{C_{n+1} - C_n}{t_{n+1} - t_n} \quad \text{en posant} \quad C = [R]$$

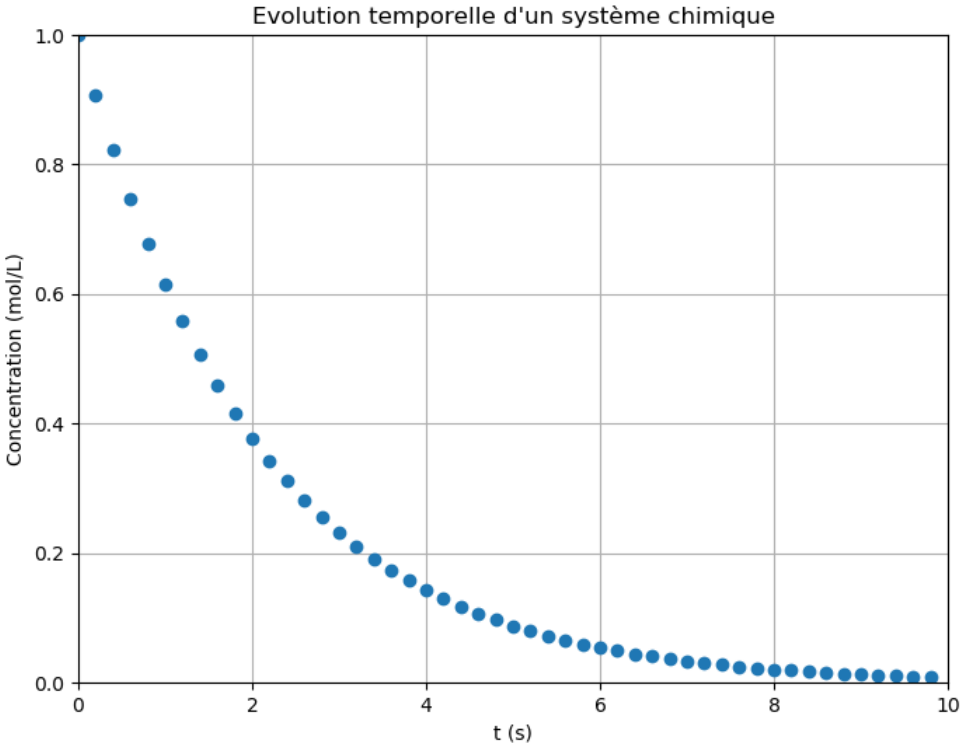
Il est donc possible de tracer l'évolution de la concentration du réactif en fonction de temps **de proche en proche** à partir de la relation :

$$C_{n+1} = C_n - k \times (t_{n+1} - t_n) \times C_n$$

Programme Python

```
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (8,6)  # width, height in inches (100 dpi)

Dt = 0.2
N = 50
t = [Dt*i for i in range(N)]
C = [0]*N

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, "o")
plt.title("Evolution temporelle d'un système chimique")
plt.xlabel("t (s)")
plt.xlim(0, 10)
plt.ylabel("Concentration (mol/L)")
plt.ylim(0, 1)
plt.grid()
plt.show()
```

Résultats



Animation Le programme suivant trace les points de la courbe pas à pas en mettant en évidence à chaque fois la concentration restante en solution et le vecteur vitesse de disparition ($v_{disp} = -k \times C_n$).

Ici une boucle *while* (avec une condition sur la concentration) est utilisée à la place d'un *for* !

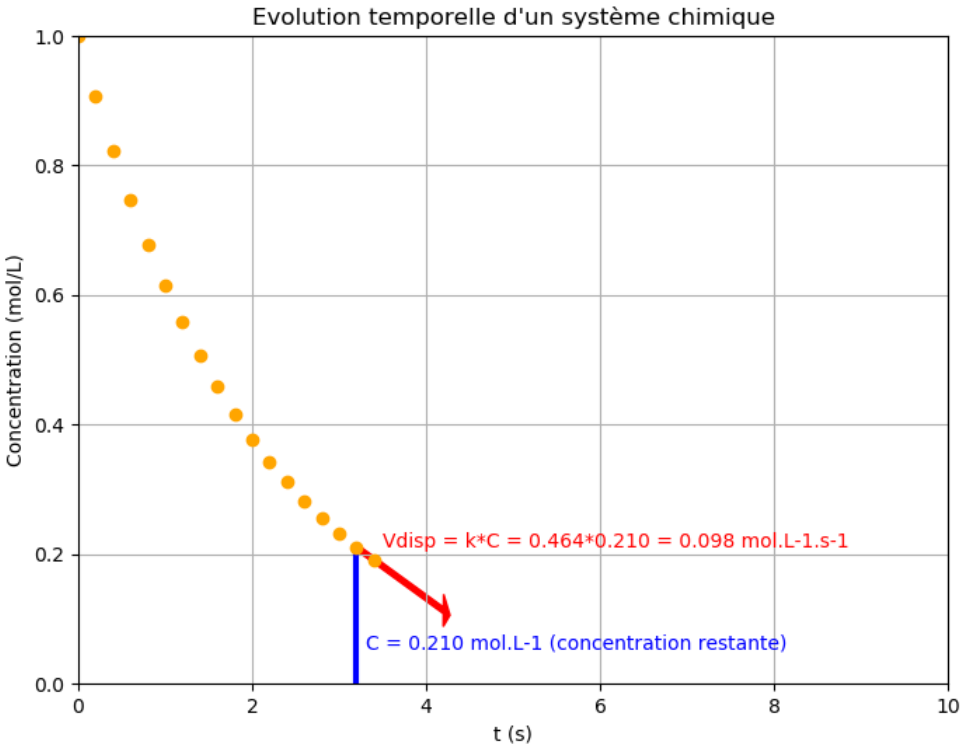
```
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (8,6)  # width, height in inches (100 dpi)

Dt = 0.2    # Pas temporel
k = 0.464   # Constante de vitesse

t = [0]     # Tableau du temps initialisé à 0
C = [1.000] # Tableau des concentrations initialisé à Co
Clim = 0.2  # Concentration qui arrête la boucle while



while C[-1] > Clim :
   # Calculs
   a = k*C[-1]                              # Calcul de la vitesse
   t.append(t[-1]+Dt)                       # Ajoute tn+1 dans le tableau des temps
   C.append(C[-1] - (t[-1]-t[-2])*k*C[-1]) # Calcul de Cn+1 et ajout dans tableau des␣
↪concentrations

   # Efface la figure
   plt.clf()

   # Tracé de la concentration restante
   plt.plot([t[-2], t[-2]], [0, C[-2]], color='blue', linewidth=3)
```

(suite sur la page suivante)

(suite de la page précédente)

```
    plt.text(t[-2]+0.1, C[-2]/4, "C = {:.3f} mol.L-1 (concentration restante)".format(C[-
↪2]), color='blue')

    # Tracé du vecteur vitesse (tangente)
    plt.arrow(t[-2], C[-2], 1, -a ,color='red', width=0.01, head_width=0.05)
    plt.text(t[-2]+0.3, C[-2], "Vdisp = k*C = {:.3f}*{:.3f} = {:.3f} mol.L-1.s-1".
↪format(k, C[-2], k*C[-2]), color='red')

    # Tracé des points de la courbe
    plt.plot(t, C, "o", color="orange")

    # Autres
    plt.title("Evolution temporelle d'un système chimique")
    plt.xlabel("t (s)")
    plt.xlim(0, 10)
    plt.ylabel("Concentration (mol/L)")
    plt.ylim(0, 1)
    plt.grid()

    # Pause
    plt.pause(2)

#plt.savefig("evolution_temporelle_disparition_anim.png")
plt.show()
```

Résultats



5.3.5 Taux d'avancement final d'une transformation chimique

Programme de classe terminale, enseignement de spécialité, voie générale

Déterminer, à l'aide d'un langage de programmation, le taux d'avancement final d'une transformation, modélisée par la réaction d'un acide sur l'eau.

5.3.5.1 Principe

La réaction d'un acide sur l'eau est décrite par la réaction suivante :

$$AH_{(aq)} + H_2O_{(l)} = A^-_{(aq)} + H_3O^+_{(aq)}$$

Le quotient de réaction s'écrit :

$$Q_r = \frac{[A^-] \cdot [H_3O^+]}{[AH] \cdot C_0}$$

avec

$$[A^-] = [H_3O^+] = \frac{x}{V} \qquad [AH] = \frac{n-x}{V} \qquad C_0 = 1 \text{ mol} \cdot \text{L}^{-1}$$

De plus, la constante d'acidité du couple AH/A^- en fonction de pK_a s'écrit :

$$K_a = 10^{-pK_a}$$

Lors de l'évolution du système chimique, le quotient de réaction Q_r va tendre vers la constante d'acidité K_a.

L'avancement final x_f peut se déterminer par :

— une **résolution numérique d'une équation du second degré** obtenue pour $Q_{r,f} = K_a$.
— une **évolution itérative (pas à pas) du système chimique** tant que $Q_{r,f} \leq K_a$.

Le **taux d'avancement final** s'obtient par l'expression :

$$\boxed{\tau = \frac{x_f}{max}}$$

5.3.5.2 Exemple 1 : évolution pas à pas du système chimique

Le quotient de réaction

$$Q_r = \frac{[A^-] \cdot [H_3O^+]}{[AH] \cdot C_0}$$

tend vers la constante d'acidité K_a lors de l'évolution du système chimique.

Dans le programme ci-dessous, la boucle `while` est interrompue à l'équilibre chimique donc pour l'avancement $x = x_f$.

```
# DONNEES
pKa = 4.76  # 4.76    Pour l'acide éthanoïque
n = 0.10    # (mol)   Quantité de matière d'acide versé
V = 2.00    # (L)     Volume d'eau
C0 = 1      # (mol/l) Concentration standard

# CALCULS
```

(suite sur la page suivante)

(suite de la page précédente)

```
Ka = 10**(-pKa) # Constante d'acidité
x = 0           # Initialisation avancement
dx = 0.00001    # Pas d'avancement
Qr = 0          # Quotient de réaction initial

while Qr<=Ka:
   x = x + dx               # Incrémentation avancement de dx
   C_AH = (n-x)/V           # Décrémentation de la concentration AH
   C_A = x/V                # Incrémentation de la concentration A-
   C_H3O = x/V              # Incrémentation de la concentration H3O+
   Qr = C_A*C_H3O/(C_AH*C0) # Calcul du nouveau quotient de réaction

x_f = x             # xf
x_max = n           # xmax
tau = 100*x_f/x_max # tau d'avancement final

# RESULTATS
print("x_f =", x_f)
print("tau =", tau)
print("Pour ", n, "mol d'acide ( Pka =", pKa, ") versé dans", V, "L d'eau")
print("Le taux d'avancement final est de", round(tau,2), "%")
```

Résultats

```
>>> %Run script.py
x_f = 0.0018500000000000042
tau = 1.850000000000004
Pour  0.1 mol d'acide ( Pka = 4.76 ) versé dans 2.0 L d'eau
Le taux d'avancement final est de 1.85 %
```

5.3.5.3 Exemple 2 : résolution numérique

A l'état final :

$$K_a = \frac{[A^-]_{eq} \cdot [H_3O^+]_{eq}}{[AH]_{eq} \cdot C_0} = \frac{\frac{x_f^2}{V^2}}{\frac{n-x_f}{V} \cdot C_0} = \frac{x_f^2}{(n-x_f) \cdot V \cdot C_0}$$

On obtient le polynôme du second degré suivant :

$$\boxed{x_f^2 + K_a V C_0 \, x_f - K_a n V C_0 = 0} \qquad \text{avec} \qquad C_0 = 1 \text{ mol} \cdot \text{L}^{-1}$$

dont la résolution donne x_f connaissant n, V et $K_a = 10^{-pK_a}$.

Le programme Python ci-dessous fait appel à la classe `poly1d` de la librairie `numpy` pour déterminer les racines du polynôme du seconde degré.

```
import numpy as np

def avancement_final(pKa, n, V, C0 = 1):
   """ Retourne l'avancement final de la réaction
   d'un acide sur l'eau.
   """
   Ka = 10**(-pKa)                  # Calcul de Ka
```

(suite sur la page suivante)

(suite de la page précédente)

```
    a, b, c = 1, Ka*V*C0, -Ka*n*V*C0 # Coeff. du polynome
    polynome = np.poly1d([a,b,c])    # Création du polynome
    solutions = polynome.roots       # Solutions
    return solutions[1]              # Retourne la deuxième solution

# DONNEES
pKa = 4.76  # 4.76  Pour l'acide éthanoïque
n = 0.10    # (mol) Quantité de matiere d'acide versé
V = 2.00    # (L)   Volume d'eau

# CALCULS
x_max = n                           # (mol) Avancement maximal
x_f = avancement_final(pKa, n, V) # (mol) Avancement final
tau = 100*x_f/x_max                 # (%)   Tau d'avancement final

# Affichage
print("x_f =", x_f)
print("tau =", tau)
print("Pour ", n, "mol d'acide ( Pka =", pKa, ") versé dans", V, "L d'eau")
print("Le taux avancement final est de", round(tau,2), "%")
```

Résultats

```
>>> %Run script.py
x_f = 0.0018469995427119795
tau = 1.8469995427119794
Pour  0.1 mol d'acide ( Pka = 4.76 ) versé dans 2.0 L d'eau
Le taux avancement final est de 1.85 %
```

5.3.6 Diagramme de distribution des espèces d'un couple acide-base

Programme de classe terminale, enseignement de spécialité, voie générale

Tracer, à l'aide d'un langage de programmation, le diagramme de distribution des espèces d'un couple acide-base de pKa donné.

5.3.6.1 Théorie

Soit un couple acide/base de la forme AH/A^- respectant la réaction suivante à l'eau :

$$AH_{(aq)} + H_2O_{(l)} = A^-_{(aq)} + H_3O^+_{(aq)}$$

La **proportion d'acide à l'équilibre** s'écrit :

$$p_{AH} = \frac{[AH]_{eq}}{[AH]_{eq} \cdot [A^-]_{eq}} = \frac{1}{1 + \frac{[AH]_{eq}}{[A^-]_{eq}}}$$

De plus :

$$pH = pK_a + \log(\frac{[A^-]_{eq}}{[AH]_{eq}}) \quad \Longrightarrow \quad \frac{[A^-]_{eq}}{[AH]_{eq}} = 10^{pH-pK_a}$$

D'où :

$$\boxed{p_{\text{AH}} = \frac{1}{1+10^{pH-pK_a}}}$$

Puis la **proportion de base à l'équilibre** :

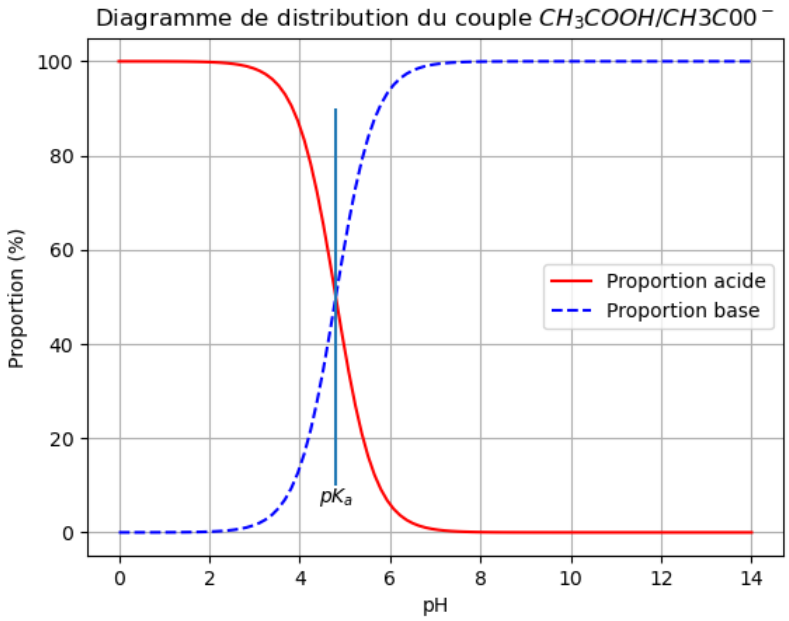
$$\boxed{p_{\text{A}^-} = 1 - p_{\text{AH}}} \quad \Longrightarrow \quad p_{\text{A}^-} = \frac{10^{pH-pK_a}}{1+10^{pH-pK_a}}$$

5.3.6.2 Programmes Python

Exemple du couple CH_3COOH/CH_3COO^- (acide éthanoïque/ion éthanoate).

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


pH = np.linspace(0, 14, 100)
pKa = 4.8  # pKa du couple acide/base

p_acide = 1/(1 + 10**(pH-pKa))
p_base = 1 - p_acide


plt.plot(pH, 100*p_acide, 'r-', label = 'Proportion acide')
plt.plot(pH, 100*p_base, 'b--', label = 'Proportion base')

plt.vlines(pKa, 10, 90)   # pKa
plt.text(pKa, 10, r"$pK_a$", ha = 'center', va = 'top')

plt.title(r"Diagramme de distribution du couple $CH_3COOH/CH3COO^-$")
plt.legend()
plt.ylabel('Proportion (%)')
plt.xlabel('pH')
plt.grid()
plt.show()
```

Résultat

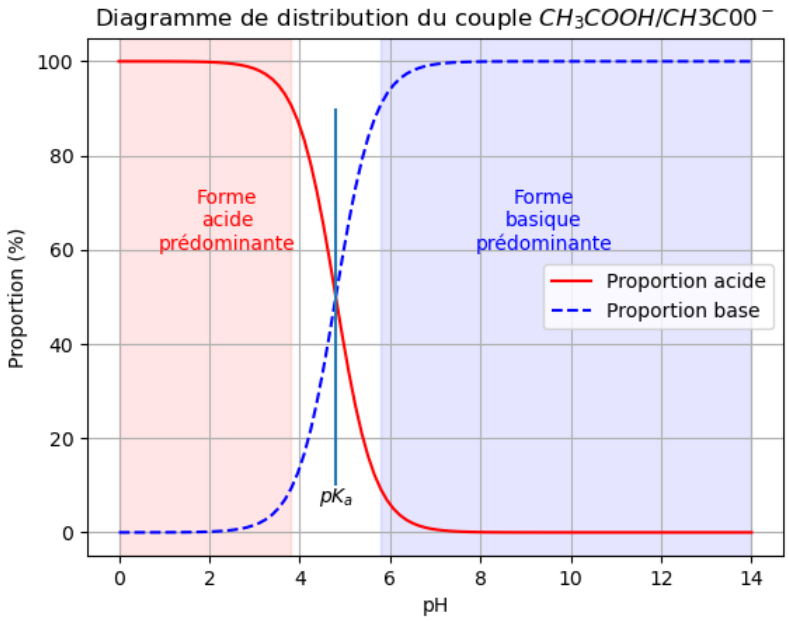


Puis en mettant en évidence les parties où :

— la **forme acide est prédominante** pour $pH < pK_a - 1$;
— la **forme basique est prédominante** pour $pH > pK_a + 1$.

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


pH = np.linspace(0, 14, 100)
pKa = 4.8  # pKa du couple acide/base

p_acide = 1/(1 + 10**(pH-pKa))
p_base = 1 - p_acide


plt.plot(pH, 100*p_acide, 'r-', label = 'Proportion acide')
plt.plot(pH, 100*p_base, 'b--', label = 'Proportion base')

plt.vlines(pKa, 10, 90)   # pKa
plt.text(pKa, 10, r"$pK_a$", ha = 'center', va = 'top')

plt.axvspan(0, pKa-1, color='r', alpha=0.1)   # Zone forme acide prédominante
plt.text(pKa/2, 60, "Forme\nacide\nprédominante", color='r', ha='center')

plt.axvspan(pKa+1, 14, color='b', alpha=0.1)   # Zone forme acide prédominante
plt.text((14+pKa)/2, 60, "Forme\nbasique\nprédominante", color='b', ha='center')

plt.title(r"Diagramme de distribution du couple $CH_3COOH/CH3C00^-$")
plt.legend()
plt.ylabel('Proportion (%)')
plt.xlabel('pH')
plt.grid()
plt.show()
```

Résultat



5.3.7 Vecteurs accélération d'un point en mouvement

Programme de classe terminale, enseignement de spécialité, voie générale

Représenter, à l'aide d'un langage de programmation, des vecteurs accélération d'un point lors d'un mouvement.

5.3.8 Evolution des énergies d'un système en mouvement dans un champ uniforme

Programme de classe terminale, enseignement de spécialité, voie générale

Représenter, à partir de données expérimentales variées, l'évolution des grandeurs énergétiques d'un système en mouvement dans un champ uniforme à l'aide d'un langage de programmation ou d'un tableur.

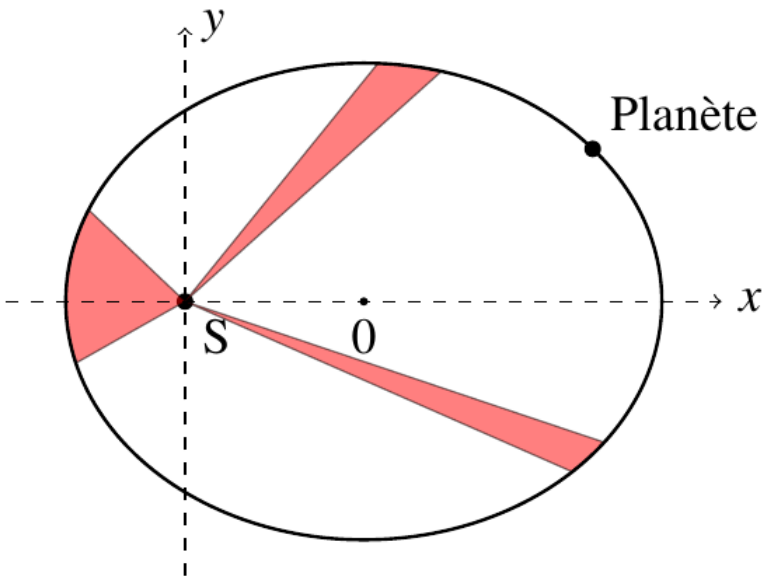
5.3.9 Seconde loi de Kepler

Programme de classe terminale, enseignement de spécialité, voie générale

Exploiter, à l'aide d'un langage de programmation, des données astronomiques ou satellitaires pour tester les deuxième et troisième lois de Kepler.

5.3.9.1 Principe

Soit une planète en orbite elliptique autour du soleil dans le référentiel héliocentrique supposé galiléen.



La seconde loi de Kepler suppose que **les aires balayées sont égales pendant des durées égales**.

La **formule de Heron** donne une estimation de cette aire à partir d'une forme triangulaire.

$$A = \sqrt{p(p-L_1)(p-L_2)(p-L_3)} \qquad \text{avec} \qquad p = \frac{L_1+L_2+L_3}{2}$$

5.3.9.2 Exemple : cas de Neptune

Dans cet exemple, les positions `x` et `y` (en ua) de Neptune par rapport au soleil sont données pour les années 2020 à 2024.

```
import matplotlib.pyplot as plt
from math import sqrt

# Positon de Neptune
t = [2020, 2021, 2022, 2023, 2024] # années
x = [29.4328410560730, 29.6118356847488, 29.7457682031831, 29.8342404541355, 29.
 ↪8774316380940] # en ua
y = [-5.3906071859853, -4.2505317595919, -3.1034908723970, -1.9519995345194, -0.
 ↪7985426911486] # en ua

def aire_balayee(x1, y1, x2, y2):
    """ Retourne l'aire balayée entre deux points par la méthode de Heron
    """
    L1 = sqrt(x1**2 + y2**2)
    L2 = sqrt(x2**2 + y2**2)
    L3 = sqrt((x2-x1)**2 + (y2-y1)**2)
    p = (L1+L2+L3)/2
    return sqrt(p*(p-L1)*(p-L2)*(p-L3))

Dt = []
aire = []
for i in range(len(x)-1):
    Dt.append(t[i+1]-t[i])
    aire.append(aire_balayee(x[i], y[i], x[i+1], y[i+1]))

print("Delta t (an) :\n", Dt)
print("Aire balayée (ua2) :\n", aire)
```

Résultats

```
>>> %Run script.py
Delta t (an) :
[1, 1, 1, 1]
Aire balayée (ua2) :
[17.00345082200997, 17.112266685090646, 17.185248384788228, 17.22165946734367]
```

On vérifie que les aires balayées sont pratiquement égales !

5.3.10 Troisième loi de Kepler

Programme de classe terminale, enseignement de spécialité, voie générale

Exploiter, à l'aide d'un langage de programmation, des données astronomiques ou satellitaires pour tester les deuxième et troisième lois de Kepler.

5.3.10.1 Principe

Vérifier la troisième loi de Kepler :

$$\boxed{\frac{T^2}{a^3} = \text{constante}} \qquad \text{ou} \qquad \frac{T^2}{r^3} = \frac{4\pi^2}{G \cdot M_S}$$

5.3.10.2 Exemple : satellites de Jupiter

Le tableau ci-dessous donne les valeurs du demi-grand axe de l'orbite et la période de révolution de quelques satellites de Jupiter.

Satellites	Amalthee	Thébé	Io	Europe	Ganymède	Callisto
a (10^3 km)	181	221	421	671	1070	1880
T (J)	0,498	0,674	1,769	3,551	7,155	16,689

Le programme ci-dessous réalise une régression linéaire afin de vérifier la proportionnalité entre T^2 et r^3.

```
from scipy.stats import linregress

# DONNEES
s = ['Amalthee', 'Thébé', 'Io', 'Europe', 'Ganymède', 'Callisto']  # Nom des satellites
r = np.array([1.81E5, 2.21E5, 4.21E5, 6.71E5, 1.07E6, 1.88E6])     # (km) Rayon orbite
T = np.array([0.498 , 0.674 , 1.769 , 3.551 , 7.155 , 16.689])     # (J)  Période de␣
 ↪révolution

# CALCULS
r3 = r**3
T2 = T**2

# MODELISATIOM PAR REGRESSION LINEAIRE
(a, b, _, _, _) = linregress(r3, T2)
print("a =", a)
print("b =", b)
T2_model = a*r3
```

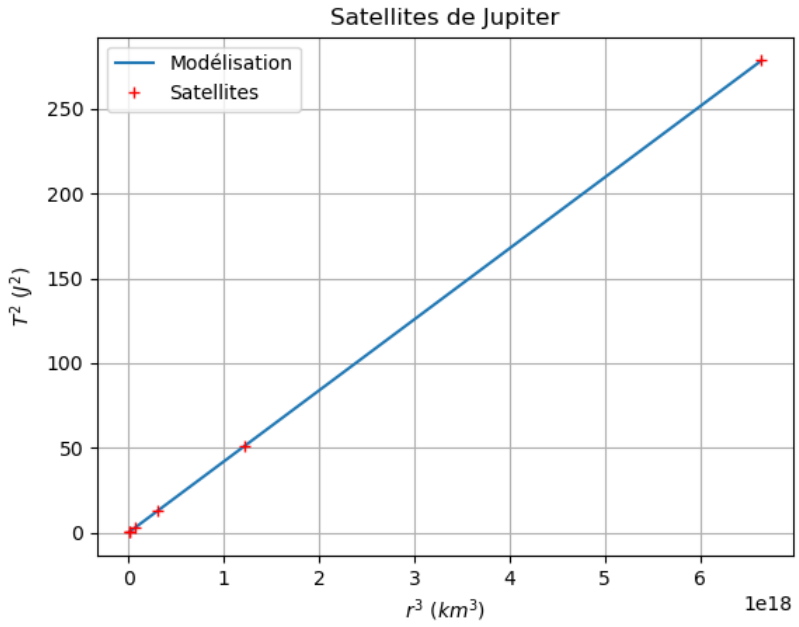
(suite sur la page suivante)

(suite de la page précédente)

```
# COURBE
plt.plot(r3, T2_model, label="Modélisation")
plt.plot(r3, T2, 'r+', label="Satellites")
plt.legend()
plt.title("Satellites de Jupiter")
plt.xlabel(r"$r^3~(km^3)$")
plt.ylabel(r"$T^2~(J^2)$")
plt.grid()
plt.savefig("periode_de_jupiter.png")
plt.show()
```

Résultats

```
>>> %Run periode_de_jupiter.py
a = 4.191896436656507e-17
b = -0.03758156536986945
```



5.3.11 Interférence de deux ondes lumineuse

Programme de classe terminale, enseignement de spécialité, voie générale

Représenter, à l'aide d'un langage de programmation, la somme de deux signaux sinusoïdaux périodiques synchrones en faisant varier la phase à l'origine de l'un des deux.

Chapitre 6

Aller plus loin

6.1 Résolution d'une équation différentielle avec la méthode d'Euler implicite (CPGE)

6.1.1 Équation différentielle d'ordre 1

Soit l'équation différentielle de la forme suivante :

$$\tau \cdot y'(t) + y(t) = g(t)$$

qui peut également s'écrire comme suit :

$$y'(t) = \frac{g(t) - y(t)}{\tau}$$

ou plus généralement :

$$\boxed{y'(t) = F(y(t), t)} \qquad \text{avec} \qquad F(y(t), t) = \frac{g(t_n) - y(t_n)}{\tau}$$

6.1.2 Méthode d'Euler implicite

En 1770, Euler propose une approximation de la dérivée :

$$y'(t_n) \approx \frac{y(t_{n+1}) - y(t_n)}{h} \qquad \text{avec} \qquad h = t_{n+1} - t_n$$

qui donne la relation suivante :

$$y(t_{n+1}) \approx y(t_n) + h \cdot y'(t_n) \qquad \text{ou encore} \qquad \boxed{y(t_{n+1}) \approx y(t_n) + h \cdot F(y(t), t)}$$

On obtient ainsi une équation de récurrence :

$$\begin{cases} y_{n+1} \approx y_n + h \cdot F(y_n, t_n) & \text{pour} \quad n = 0, 1, ..., N \\ y_0 = y(0) \end{cases}$$

A l'instant initial $t = 0$, on a $y(0) = y_0$.

6.1.3 Implémentation d'une fonction `euler`

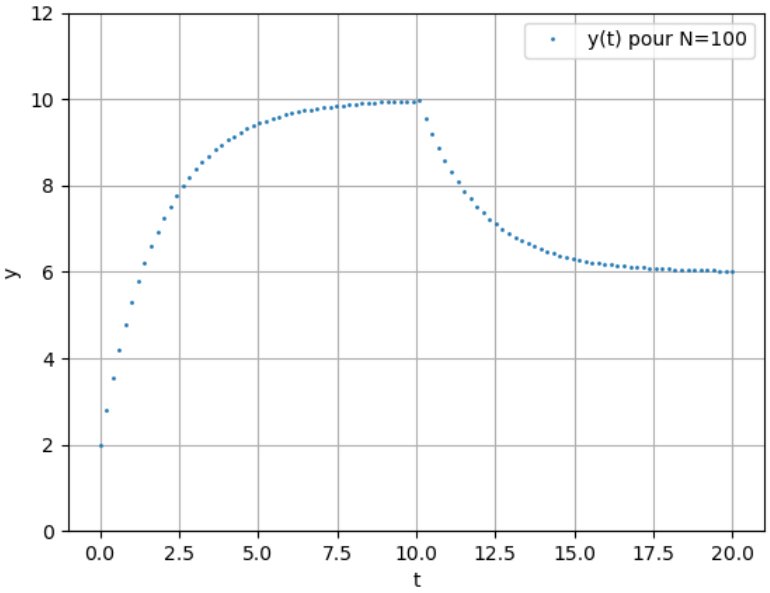
```
import numpy as np
import matplotlib.pyplot as plt

def derive(y, t):
    if t<T/2:
        g = 10
    else:
        g = 6
    return (g-y)/tau

def euler(derive, y0, t):
    h = t[1]-t[0]
    N = len(t)
    y = np.zeros(N)
    y[0] = y0
    for n in range(0,N-1):
        y[n+1] = y[n] + h*derive(y[n], t[n])
    return y

# PARAMETRES
tau = 2
T = 20
N = 1000

# RESOLUTION DE L'EQUADIFF
y0 = 2                    # Condition initiale
t = np.linspace(0,T,N)    # Tableau du temps
y = euler(derive, y0, t)  # Integration

# COURBES
plt.plot(t, y, ".", ms =2 ,label='y(t) pour N={}'.format(N))
plt.legend()
plt.xlabel('t')
plt.ylim(0,12)
plt.ylabel('y')
plt.grid()
plt.show()
```

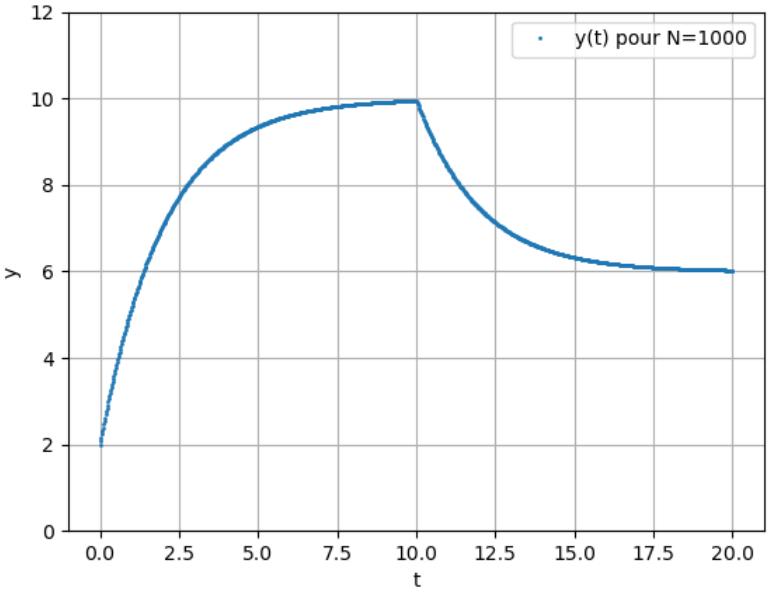
La fonction d'intégration `euler` :

- — détermine le pas `h` et le nombre de points `N`.
- — pose la condition initiale `y[0] = y0`.
- — calcule les termes `y[n+1]` par récurrence dans une boucle de `N` itérations.

La fonction `derive` caractérise ici l'équation différentielle prise comme exemple.

Résultats





6.1.4 Avec la fonction `odeint`

La fonction `odeint` de `scipy.integrate` utilise une autre méthode d'intégration plus précise que celle d'Euler.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

def derive(y, t):
    if t<T/2:
        g = 10
```

(suite sur la page suivante)

(suite de la page précédente)

```
    else:
        g = 6
    return (g-y)/tau

# PARAMETRES
tau = 2
T = 20
N = 100

# RESOLUTION DE L'EQUADIFF
y0 = 2                          # Condition initiale
t = np.linspace(0,T,N)          # Tableau du temps
y = odeint(derive, y0, t)

# COURBES
plt.plot(t, y, ".", ms =2 ,label='y(t) pour N={}'.format(N))
plt.legend()
plt.xlabel('t')
plt.ylim(0,12)
plt.ylabel('y')
plt.grid()
plt.show()
```

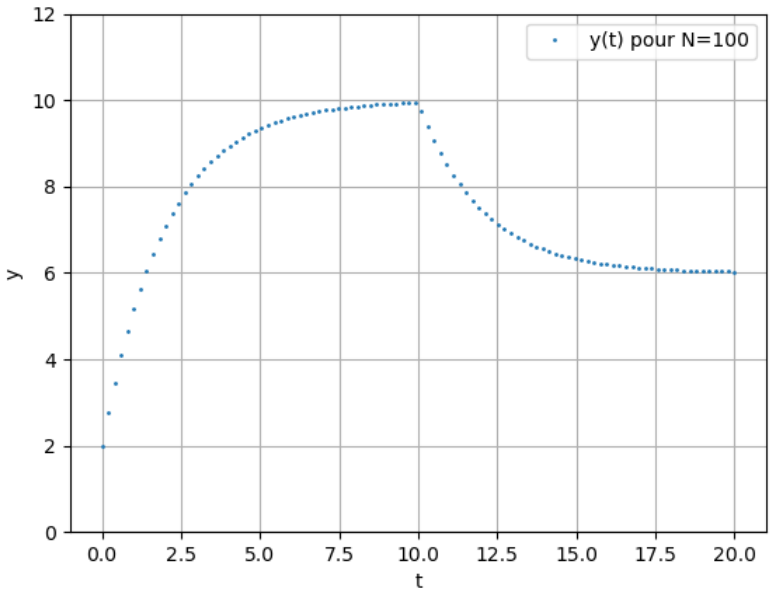


FIG. 1 – $N = 100$

Chapitre 7

Annexes

7.1 Installation de paquets Python à travers un Proxy

7.1.1 Qu'est-ce qu'un Proxy ?

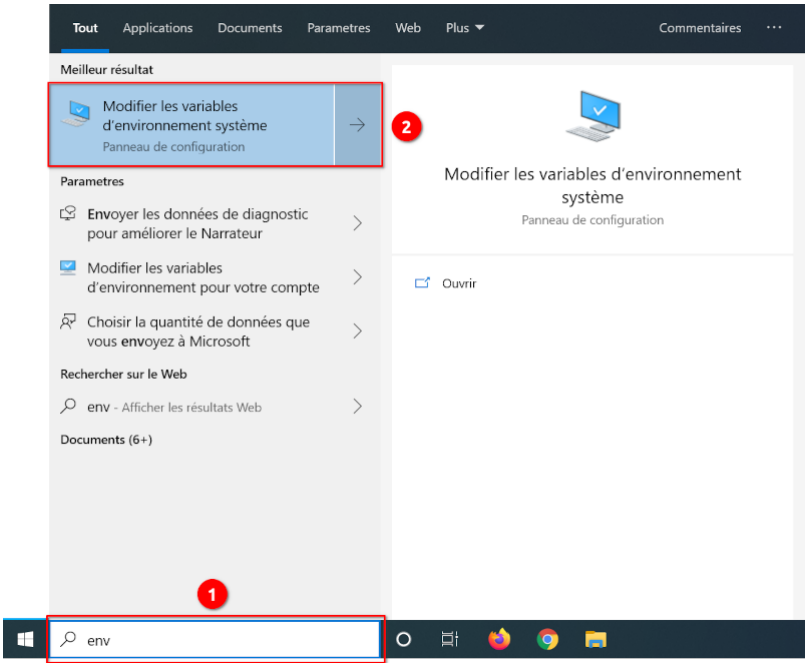
Dans la majorité des établissements scolaires, l'accès à Internet se fait par l'intermédiaire d'une composante logicielle appelée **Proxy**. Il permet notamment d'**accélérer la navigation Web** (mémoire cache) mais surtout d'**authentifier les utilisateurs** qui veulent accéder à Internet.

— Un **serveur Proxy** est référencé par une **adresse IP** (ex. `172.16.8.1`) et un **numéro de port** (ex. `3129`).
— Un utilisateur Proxy possède un **identifiant** (ex. `user`) et un **mot de passe** (ex. `passwd`).

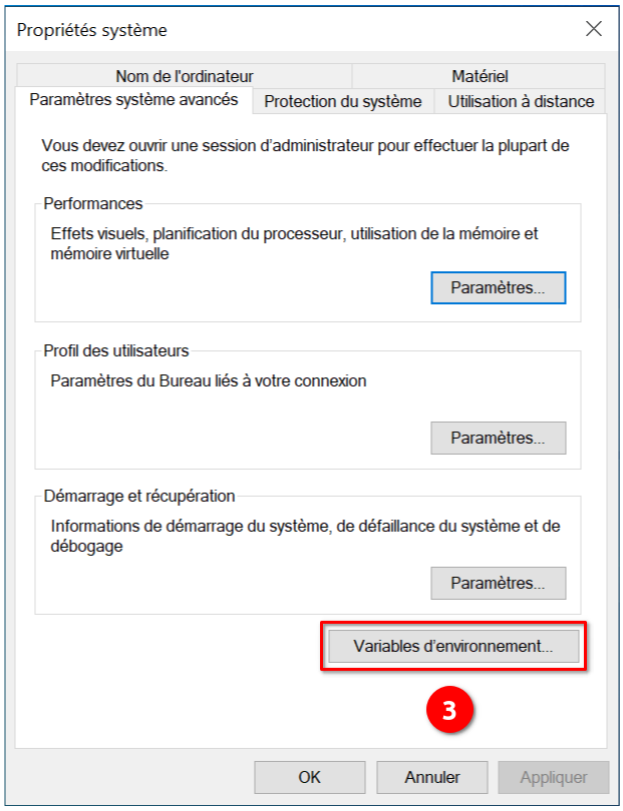
7.1.2 Paramétrage du Proxy dans Windows

L'ajout des paramètres d'authentification d'un utilisateur Proxy dans les variables d'environnement de Windows se fait suivant les étapes ci-dessous avec des **droits d'administrateur système**.

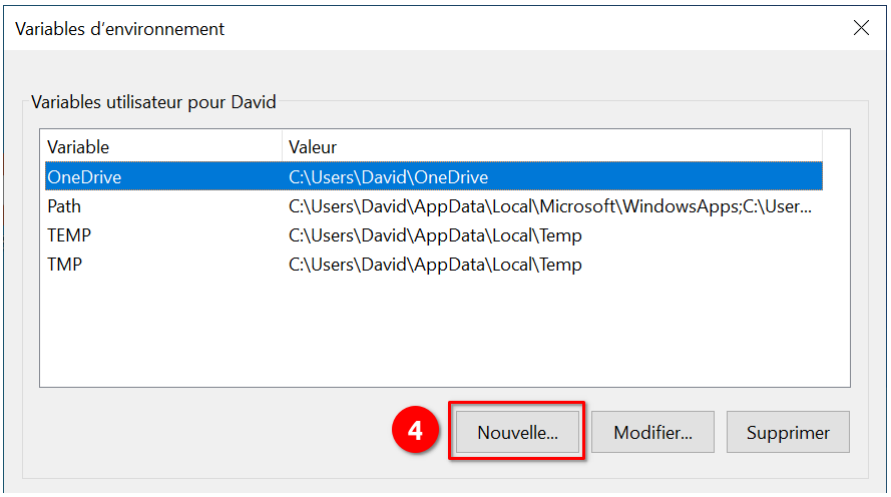
1. Chercher "Modifier les variables d'environnement système".
2. Ouvrir la fenêtre le panneau de configuration.



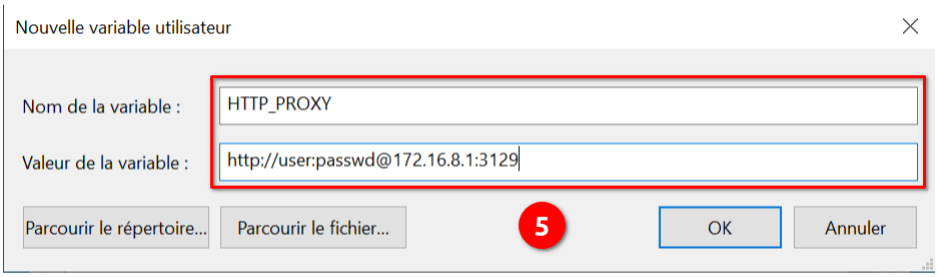
3. Accéder aux variables d'environnement.



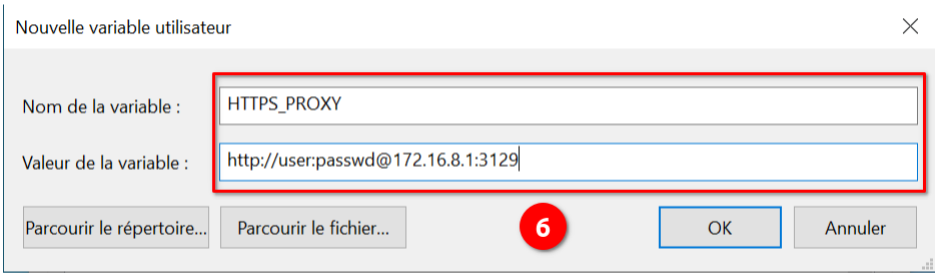
4. Ajouter une nouvelle variable d'environnement.



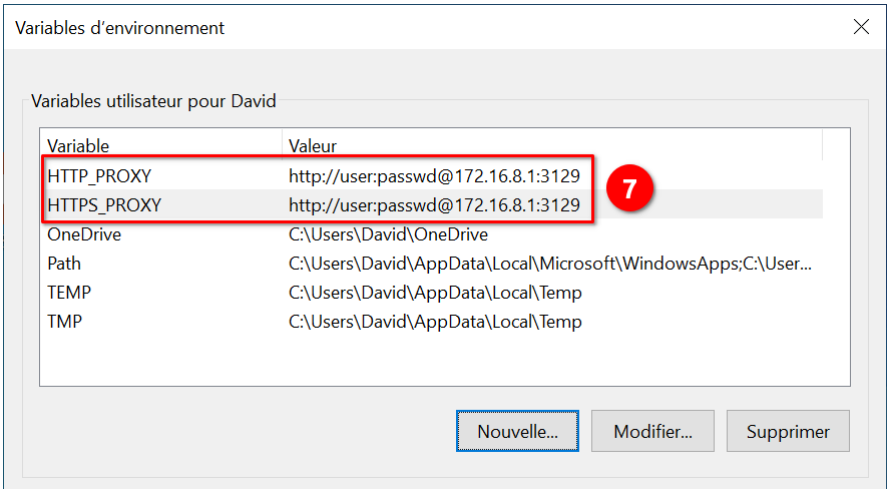
5. Saisir la variable `HTTP_PROXY` et la valeur correspondante.



6. Saisir la variable `HTTPS_PROXY` et la valeur correspondante.



7. Les deux nouvelles variables `HTTP_PROXY` et `HTTPS_PROXY` sont maintenant enregistrées.



A noter qu'une adresse du proxy s'écrit toujours sous la forme suivante :

```
http://user:passwd@172.16.8.1:3129
```

où :

— `user` est l'**identifiant** de l'utilisateur,
— `passwd` est le **mot de passe** de l'utilisateur,
— `172.16.8.1` est l'**adresse IP** du proxy,
— `3129` est le **numéro de port** du proxy.

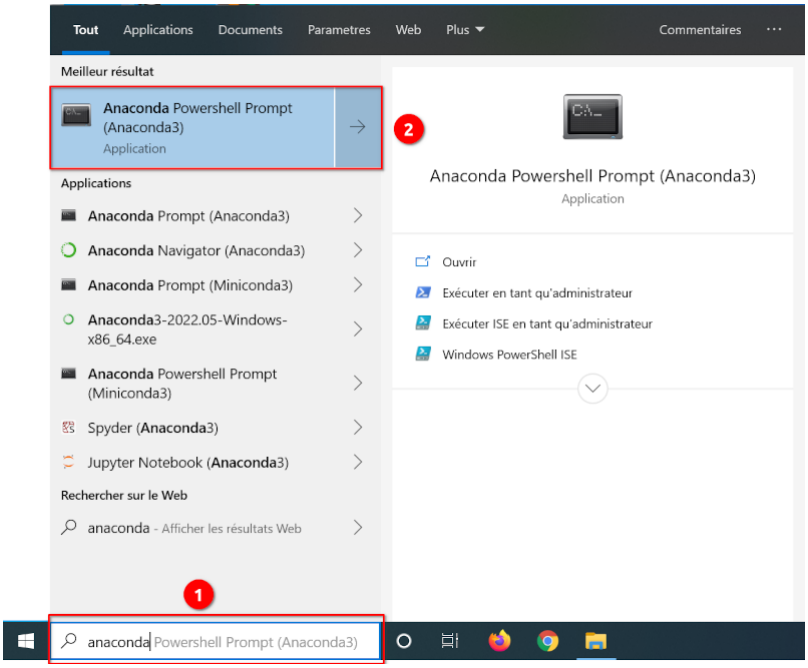
Avertissement :

— Ces quatre valeurs précédentes sont des exemples. Elles sont à modifier en fonction de la configuration du proxy de votre établissement.
— Il est conseillé d'enregistrer un **utilisateur virtuel** comme par exemple `physique.windows` pour des raisons de sécurité.
— **Redémarrer l'ordinateur** une fois l'opération terminée. **Accepter le certificat** si le Windows le demande.

7.1.3 Installation de paquets avec Anaconda

Comme la plupart des distributions Python, Anaconda propose l'utilitaire en ligne de commande **pip** pour l'**installation** et la **mise à jour de paquets** (librairies, modules, ...)

1. Faire une recherche à partir du mot clé "Anaconda".
2. Ouvrir le terminal **Anaconda Powershell Prompt**.

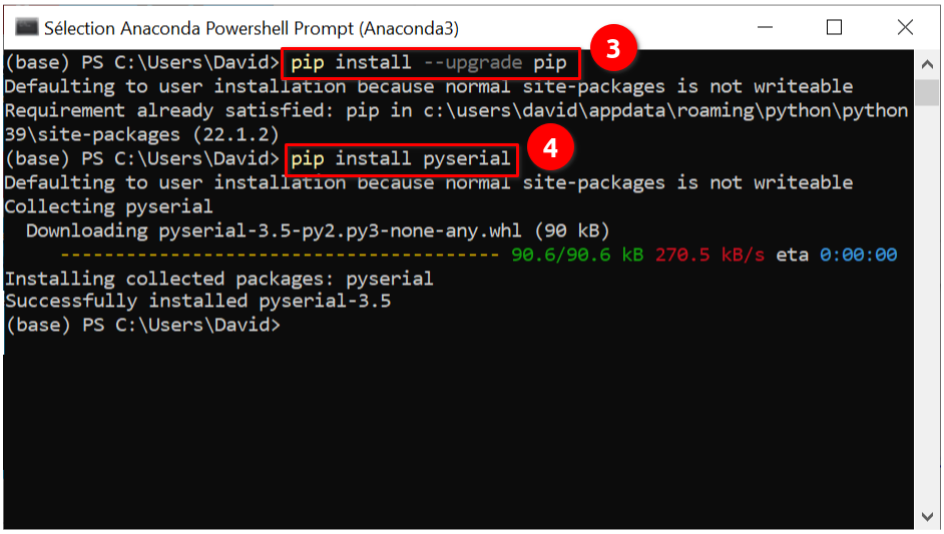


3. Mettre à jour l'utilitaire **pip** avec la commande :

```
pip install --upgrade pip
```

4. Par exemple, pour installer le module `pyserial` (accès au port série) :

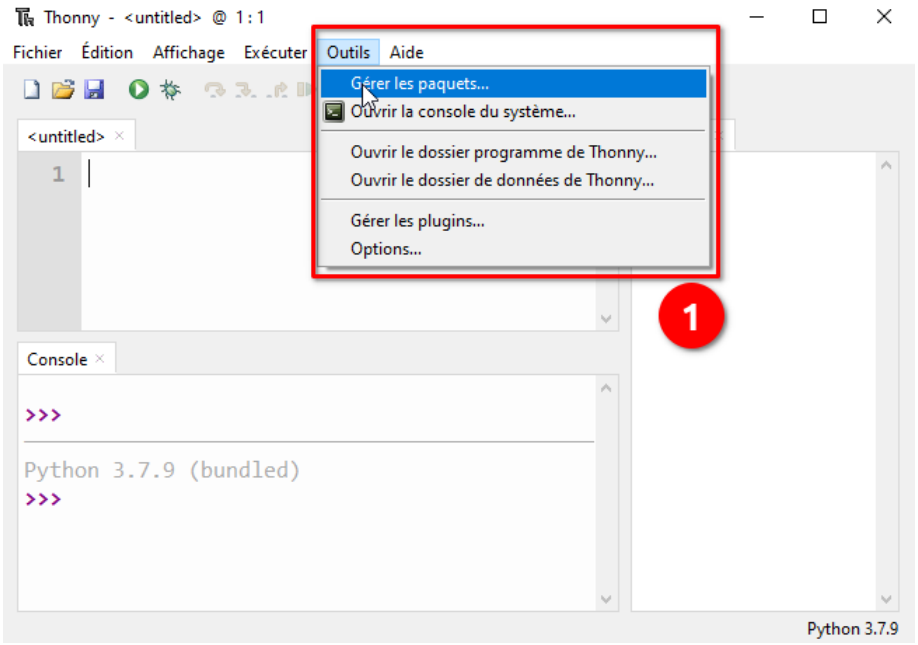
```
pip install pyserial
```



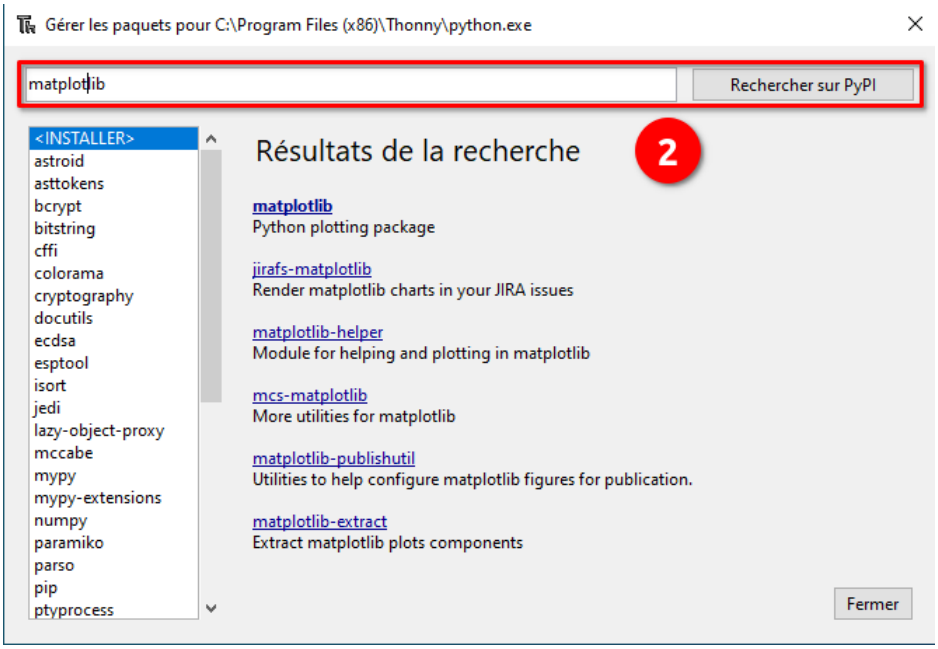
7.1.4 Installation de paquets avec Thonny

Thonny utilise également **pip** en arrière plan pour la gestion des paquets.

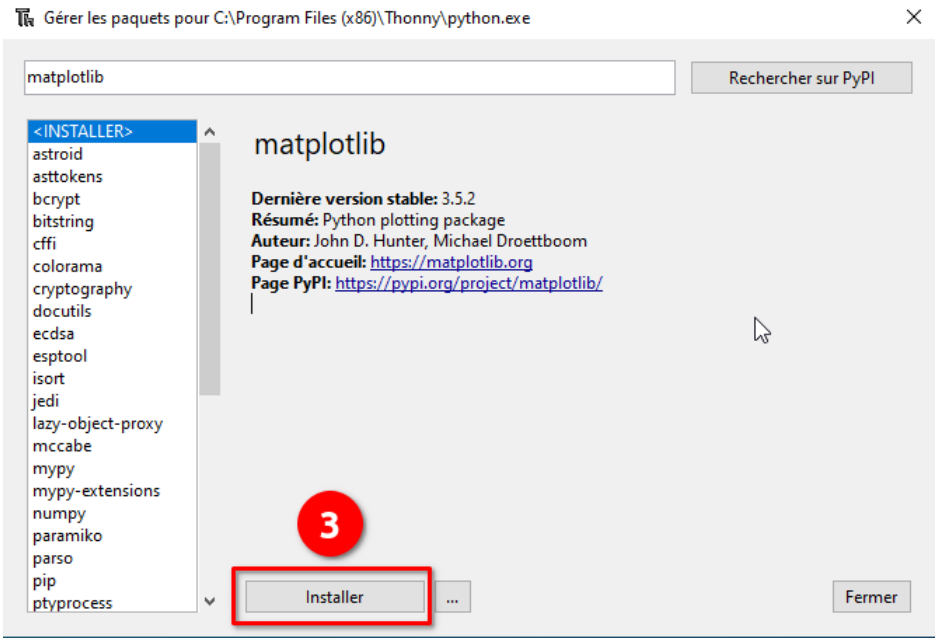
1. Ouvrir la fenêtre de gestion des paquets.



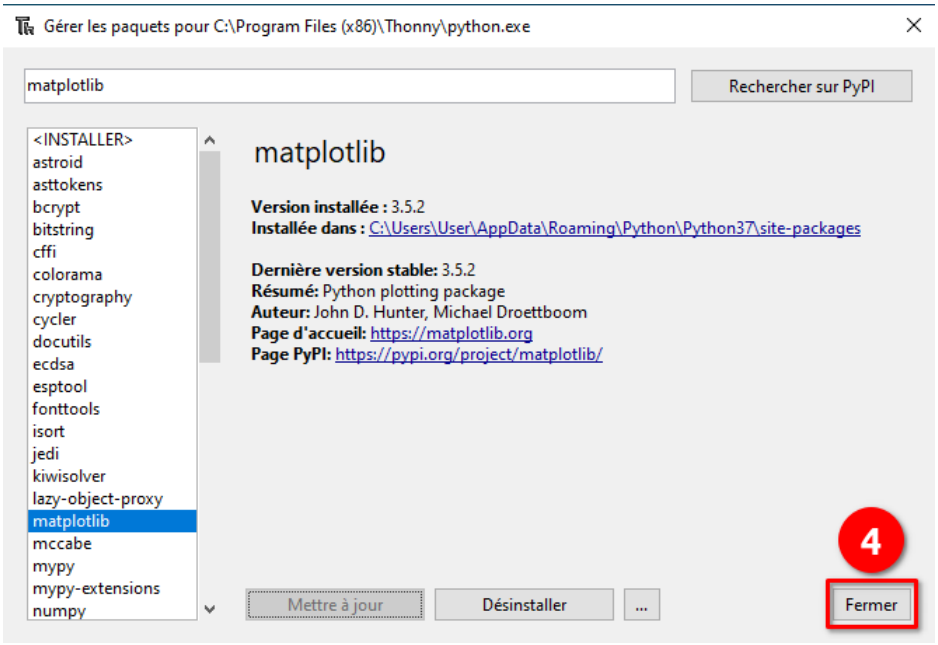
2. Rechercher le paquet (ex. matplotlib)



3. Un fois trouvé et sélectionné, installer le paquet.



4. L'installation est terminée.



7.2 Correction des exercices

Exercice 1. (dilution)

On sait que :

$$F = \frac{C_m}{C_f} \qquad \text{et} \qquad V_m = \frac{V_f}{F}$$

```
Cm =  30  # (g/L) concemtration massique solution mère
Cf =  10  # (g/L) concentration massique solution fille
Vf = 500  # (mL)  Volume solution fille

F = Cm/Cf        # (--) Calcul du facteur de dilution
Vm = Vf/F        # (mL) Calcul de Vm
Veau = Vf-Vm     # (mL) Calcul de Veau (volume d'eau à compléter)


# AFFICHAGE
print("Prélever ", round(Vm,0), " mL de solution mère.")
print("Puis ajouter", round(Veau,0), "mL d'eau pour obtenir", round(Vf,0), "mL de␣
 ↪solution fille")
print("Le facteur de dilution est de", round(F,2))
```

Ou avec l'affichage du texte avec la méthode `format` du type `str`.

```
Cm =  30  # (g/L) concemtration massique solution mère
Cf =  10  # (g/L) concentration massique solution fille
Vf = 500  # (mL)  Volume solution fille

F = Cm/Cf        # (--) Calcul du facteur de dilution
```

(suite sur la page suivante)

(suite de la page précédente)

```
Vm = Vf/F        # (mL) Calcul de Vm
Veau = Vf-Vm     # (mL) Calcul de Veau (volume d'eau à compléter)


# AFFICHAGE
print("Prélever {:.0f} mL de solution mère.".format(Vm))
print("Puis ajouter {:.0f} mL d'eau pour obtenir {:.0f} mL de solution fille".
 ↪format(Veau, Vf))
print("Le facteur de dilution est de {:.1f}".format(F))
```

Exercice 2. (distance focale)

```
f_prim = [19.1, 18.9, 18.7, 19.0, 18.9, 19.2, 18.8, 18.7] # Listes des mesures

nb = len(f_prim)  # Nombre de mesures

moyenne = sum(f_prim)/len(f_prim)  # Moyenne

maximum = max(f_prim)  # Maximum
minimum = min(f_prim)  # Minimum

f_prim.pop(0)   # supprime la première mesure
f_prim.pop()    # supprime la dernière mesure (-1 par défaut)

# OU
#f_prim = f_prim[1:-1] # supprime la première et la dernière
```

Exercice 3. (célérité son)

```
c = [335, 338, 341, 339, 340, 336, 343, 337, 341, 339, 337] # (m/s) célérité son

somme = 0
for i in range(len(c)):
    somme = somme + c[i]

moyenne = somme/len(c)
print("Moyenne =", moyenne)
```

Exercice 4. (vitesse)

A partir d'une liste v vide au départ.

```
t = [0. , 0.04, 0.08, 0.12, 0.16, 0.20, 0.24, 0.28, 0.32, 0.36, 0.40, 0.44, 0.48] # (s)
z = [1.66, 1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21, 0. ] # (m)

v = []
for i in range(1, len(t)-1):
    V = -(z[i+1]-z[i-1])/(t[i+1]-t[i-1])   # (m/s) Calcul la vitesse en cours
    v.append(round(V,2))                   # (m/s) Mémorisation de la vitesse

print(v)
```

```
>>> %Run
[1.62, 2.0, 2.25, 2.75, 3.13, 3.37, 3.88, 4.38, 4.62, 5.0, 5.25]
```

Attention à la taille de la liste v à la fin !

Ou à partir d'une liste v initialisée par N zéros.

```
t = [0. , 0.04, 0.08, 0.12, 0.16, 0.20, 0.24, 0.28, 0.32, 0.36, 0.40, 0.44, 0.48] # (s)
z = [1.66, 1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21, 0. ] # (m)

N = len(t)  # Taille de la liste
v = [0]*N   # Création d'une liste de 0 de taille N

for i in range(1, N-1):
    v[i] = round(-(z[i+1]-z[i-1])/(t[i+1]-t[i-1]), 2)   # (m/s) Calcul la vitesse en␣
 ↪cours

print(v)
```

```
>>> %Run
[0, 1.62, 2.0, 2.25, 2.75, 3.13, 3.37, 3.88, 4.38, 4.62, 5.0, 5.25, 0]
```

Exercice 5. (énergies)

```
z = [1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21] # (m)
v = [1.58, 1.95, 2.27, 2.69, 3.12, 3.49, 3.91, 4.38, 4.65, 4.91, 5.23] # (m/s)

Epp, Ec, Em = [], [], []  # Initialisations des tableaux

m = 55E-3  # (kg)   Masse de la balle
g = 9.81   # (m/s²) Accélération de pesanteur

for i in range(len(z)):
    EPP = m*g*z[i]              # Calcul l'énergie potentielle de pesanteur en cours
    EC = 0.5*m*v[i]**2          # Calcul l'énergie cinétique en cours
    Epp.append(round(EPP,3))    # Complète le tableau Epp
    Ec.append(round(EC,3))      # Complète le tableau Ec
    Em.append(round(EPP+EC,3)) # Complète le tableau Em

# AFFICHAGE
print("Epp (J) :", Epp)
print("Ec  (J) :", Ec)
print("Em  (J) :", Em)
```

```
>>> %Run
Epp (J) : [0.869, 0.826, 0.782, 0.728, 0.664, 0.594, 0.518, 0.426, 0.329, 0.227, 0.113]
Ec  (J) : [0.069, 0.105, 0.142, 0.199, 0.268, 0.335, 0.42, 0.528, 0.595, 0.663, 0.752]
Em  (J) : [0.937, 0.93, 0.924, 0.927, 0.931, 0.928, 0.938, 0.954, 0.924, 0.89, 0.866]
```

Exercice 6. (énergies)

```
z = [1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21] # (m)
v = [1.58, 1.95, 2.27, 2.69, 3.12, 3.49, 3.91, 4.38, 4.65, 4.91, 5.23] # (m/s)
```

(suite sur la page suivante)

(suite de la page précédente)

```
m = 55E-3  # (kg)   Masse de la balle
g = 9.81   # (m/s²) Accélération de la pesanteur

Epp = [m*g*Z for Z in z]          # Tableau des énergies potentielles de pesanteur
Ec  = [0.5*m*V**2 for V in v]     # Tableau des énergies cinétiques
Em  = [Epp[i]+Ec[i] for i in range(len(z))]  # Tableau des énergies mécaniques

# AFFICHAGE
print("Epp (J) :", Epp)
print("Ec  (J) :", Ec)
print("Em  (J) :", Em)
```

```
>>> %Run
Epp (J) : [0.8686755000000002, 0.8255115000000002, 0.7823475000000001, 0.
↪7283925000000001, 0.6636465000000001, 0.5935050000000002, 0.5179680000000001, 0.
↪4262445000000001, 0.32912550000000007, 0.22661100000000003, 0.11330550000000002]
Ec  (J) : [0.06865100000000002, 0.10456874999999999, 0.14170475, 0.19899275, 0.
↪26769600000000005, 0.33495275, 0.42042275000000007, 0.527571, 0.5946187500000001, 0.
↪66297275, 0.7522047500000002]
Em  (J) : [0.9373265000000002, 0.9300802500000002, 0.9240522500000001, 0.
↪9273852500000002, 0.9313425000000002, 0.9284577500000002, 0.9383907500000002, 0.
↪9538155000000001, 0.9237442500000002, 0.88958375, 0.8655102500000003]
```

ou

```
z = [1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21] # (m)
v = [1.58, 1.95, 2.27, 2.69, 3.12, 3.49, 3.91, 4.38, 4.65, 4.91, 5.23] # (m/s)

m = 55E-3  # (kg)   Masse de la balle
g = 9.81   # (m/s²) Accélération de la pesanteur

Epp = [m*g*z[i] for i in range(len(z))]      # Tableau des énergies potentielles de␣
↪pesanteur
Ec  = [0.5*m*v[i]**2 for i in range(len(v))] # Tableau des énergies cinétiques
Em  = [Epp[i]+Ec[i] for i in range(len(z))]  # Tableau des énergies mécaniques

# AFFICHAGE
print("Epp (J) :", Epp)
print("Ec  (J) :", Ec)
print("Em  (J) :", Em)
```

```
>>> %Run
Epp (J) : [0.8686755000000002, 0.8255115000000002, 0.7823475000000001, 0.
↪7283925000000001, 0.6636465000000001, 0.5935050000000002, 0.5179680000000001, 0.
↪4262445000000001, 0.32912550000000007, 0.22661100000000003, 0.11330550000000002]
Ec  (J) : [0.06865100000000002, 0.10456874999999999, 0.14170475, 0.19899275, 0.
↪26769600000000005, 0.33495275, 0.42042275000000007, 0.527571, 0.5946187500000001, 0.
↪66297275, 0.7522047500000002]
Em  (J) : [0.9373265000000002, 0.9300802500000002, 0.9240522500000001, 0.
↪9273852500000002, 0.9313425000000002, 0.9284577500000002, 0.9383907500000002, 0.
↪9538155000000001, 0.9237442500000002, 0.88958375, 0.8655102500000003]
```

Exercice 7. (nombre de dilutions)

```
Cm = 1.0          # (g/L) Concentration solution mère
Cf = 1.0e-3       # (g/L) Concentration solution fille

Nb = 0            # Nombre de dilution (initialisation)
while Cm>Cf:
    Nb = Nb + 1   # Incrémentation
    Cm = Cm/10    # Dilution par 10

print("Il faut réaliser", Nb, "dilutions !")
```

```
>>> %Run
Il faut réaliser 3 dilutions !
```

La boucle doit s'arrêter quand `Cm` est égale ou inférieure à `Cf` !

Exercice 8. (avancement)

```
a, b, c, d = 1, 2, 3, 2                    # coefficients stoechiométiques
n_A, n_B, n_C, n_D = 0.5, 0.5, 0, 0        # (mol) Quantités de matière initiales

x = 0           # Initialisation de l'avancement
dx = 0.01       # Pas de l'avancement

while (n_A>0 and n_B>0) :
    x = x + dx          # Incrémentation de l'avancement
    n_A = n_A - a*dx    # Décrémentation de n_A (réactif)
    n_B = n_B - b*dx    # Décrémentation de n_B (réactif)
    n_C = n_C + c*dx    # Incrémentation de n_C (produit)
    n_D = n_D + d*dx    # Incrémentation de n_D (produit)

# AFFICHAGE
print("xf = ", x)
print("n(A) = ", n_A)
print("n(B) = ", n_B)
print("n(C) = ", n_C)
print("n(D) = ", n_D)
```

```
>>> %Run
xf =  0.25000000000000006
n(A) =  0.24999999999999978
n(B) =  -1.734723475976807e-16
n(C) =  0.7500000000000004
n(D) =  0.5000000000000001
```

Exercice 9. (dilution)

```
def dilution(Cm, Cf, Vf):
    return Vf*Cf/Cm             # Calcul de Vm

def dilution2(Cm, Cf, Vf):
    Vm = Vf*Cf/Cm               # Calcul de Vm
```

(suite sur la page suivante)

(suite de la page précédente)

```
    Veau = Vf-Vm                # Calcul de Veau (volume d'eau à compléter)
    F = Cm/Cf                   # Calcul du facteur de dilution
    return Vm, Veau, F
```

```
>>> %Run
>>> dilution(30, 5, 100)
16.666666666666668
>>> Vm = dilution(30, 5, 100)
>>> Vm
16.666666666666668
>>> dilution2(30, 5, 100)
(16.666666666666668, 83.33333333333333, 6.0)
>>> Vm, Veau, F = dilution2(30, 5, 100)
>>> Vm
16.666666666666668
>>> Veau
83.33333333333333
>>> F
6.0
```

```
from math import log10

def potentiel_hydrogene(C_H3O_plus, C0 = 1):
    return -log10(C_H3O_plus)/C0

def concentration_ion_hydronium(pH):
    return 10**(-pH)
```

L'argument C0 fixé à 1 par défaut est optionnel.

```
>>> %Run
>>> potentiel_hydrogene(1E-5)
5.0
>>> potentiel_hydrogene(2E-5)
4.698970004336019
>>> potentiel_hydrogene(2E-5, C0=10)  # N'a pas de sens ici !
0.4698970004336019
>>> concentration_ion_hydronium(7)
1e-07
>>> concentration_ion_hydronium(6.8)
1.584893192461114e-07
```

Exercice 11. (énergies)

```
import numpy as np

z = np.array([1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21]) # (m)
v = np.array([1.58, 1.95, 2.27, 2.69, 3.12, 3.49, 3.91, 4.38, 4.65, 4.91, 5.23]) # (m/s)

m = 55e-3  # (kg)   Masse de la balle
```

(suite sur la page suivante)

(suite de la page précédente)

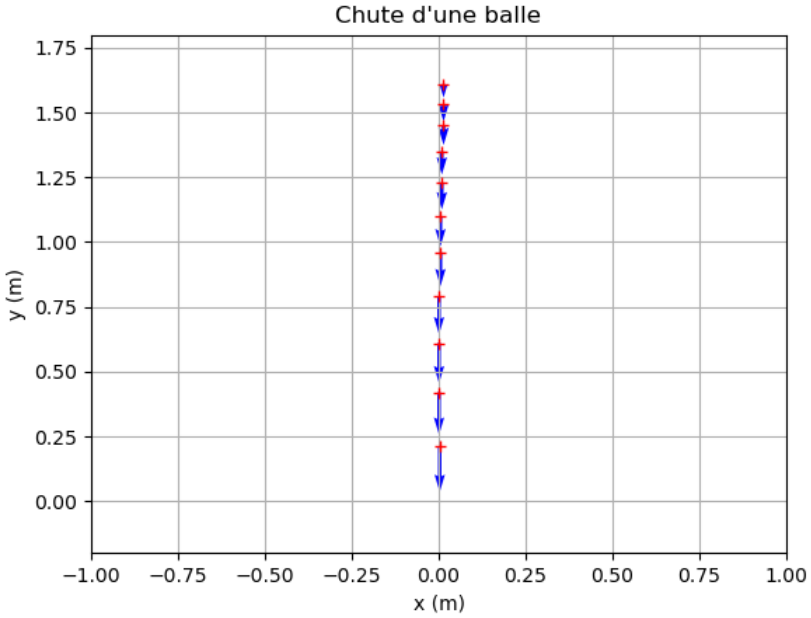
```
g = 9.81   # (m/s²) Accélération de la pesanteur

Epp = m*g*z
Ec = 0.5*m*v**2
Em = Epp + Ec

# AFFICHAGE
print("Epp (J) :", Epp)
print("Ec  (J) :", Ec)
print("Em  (J) :", Em)
```

— Les opérateurs sur les tableaux Numpy s'appliquent sur les éléments du même indice !
— Pas besoin de faire des boucles avec les tableaux Numpy

```
>>> %Run
Epp (J) : [0.8686755 0.8255115 0.7823475 0.7283925 0.6636465 0.593505  0.517968
0.4262445 0.3291255 0.226611  0.1133055]
Ec  (J) : [0.068651   0.10456875 0.14170475 0.19899275 0.267696   0.33495275
0.42042275 0.527571   0.59461875 0.66297275 0.75220475]
Em  (J) : [0.9373265  0.93008025 0.92405225 0.92738525 0.9313425  0.92845775
0.93839075 0.9538155  0.92374425 0.88958375 0.86551025]
```

Exercice 12. (positions et vecteurs vitesse)

```
import matplotlib.pyplot as plt

x = [.014, .014, .014, .009, .009, .005, .005, .001, .001, .001, .003] # (m)
z = [1.61, 1.53, 1.45, 1.35, 1.23, 1.10, 0.96, 0.79, 0.61, 0.42, 0.21] # (m)
v = [1.58, 1.95, 2.27, 2.69, 3.12, 3.49, 3.91, 4.38, 4.65, 4.91, 5.23] # (m/s)


# FIGURE
plt.plot(x, z, "r+")
for i in range(len(x)):
    plt.quiver(x[i], z[i], 0, -v[i], angles='xy', scale_units='xy', scale=30,  color=
 ↪'blue', width=0.005)
plt.title("Chute d'une balle")
plt.xlabel("x (m)")
plt.xlim(-1,1)
plt.ylabel("y (m)")
plt.ylim(-0.2,1.8)
plt.grid()
plt.show()
```



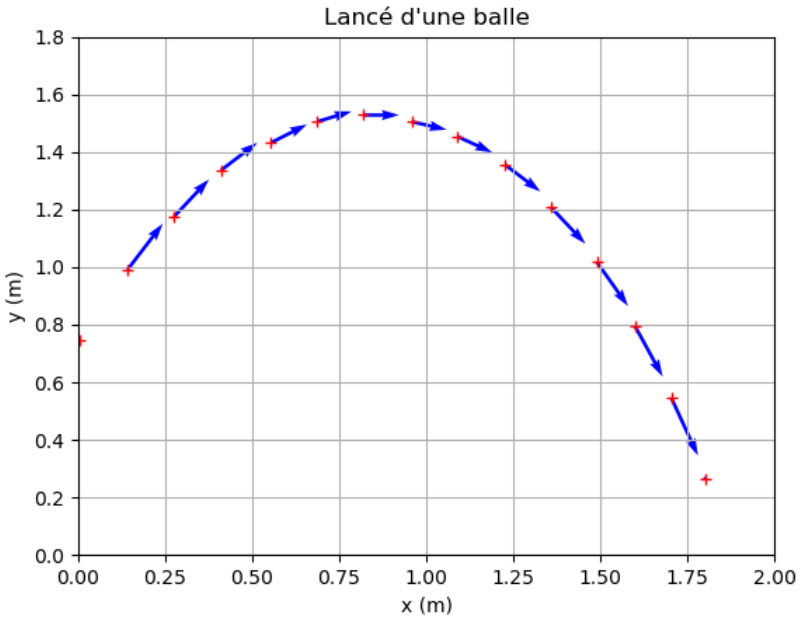
Exercice 13. (positions et vecteurs vitesse)

```
import matplotlib.pyplot as plt

# DONNEES
t = [0.0, 0.0667, 0.1334, 0.2001, 0.2668, 0.3335, 0.4002, 0.4669, 0.5336, 0.6003, 0.667,
 ↪ 0.7337, 0.8004, 0.8671, 0.9338]
x = [0.003, 0.141, 0.275, 0.410, 0.554, 0.686, 0.820, 0.958, 1.089, 1.227, 1.359, 1.490,
 ↪ 1.599, 1.705, 1.801]
y = [0.746, 0.990, 1.175, 1.336, 1.432, 1.505, 1.528, 1.505, 1.454, 1.355, 1.207, 1.018,
 ↪ 0.797, 0.544, 0.266]

# INITIALISATION
N = len(t)          # Taille
vx = [0]*N          # Liste de N zéros
vy = [0]*N          # Liste de N zéros

# CALCULS
for i in range(1, N-1):
    vx[i] = (x[i+1]-x[i-1])/(t[i+1]-t[i-1])  # Calcul de la composante vx
    vy[i] = (y[i+1]-y[i-1])/(t[i+1]-t[i-1])  # Calcul de la composante vy

# AFFICHAGE
print("vx :", vx)
print("vy :", vx)

# FIGURE
plt.plot(x, y, "r+")
for i in range(1, N-1):
    plt.quiver(x[i], y[i], vx[i], vy[i], angles='xy', scale_units='xy', scale=20, ␣
 ↪color='blue', width=0.005)
plt.title("Lancé d'une balle")
plt.xlabel("x (m)")
plt.xlim(0,2)
```

(suite sur la page suivante)

(suite de la page précédente)

```
plt.ylabel("y (m)")
plt.ylim(0,1.8)
plt.grid()
plt.show()
```




```
>>> %Run Exo13.py
vx : [0, 2.038980509745128, 2.0164917541229386, 2.0914542728635683, 2.0689655172413794,␣
 ↪1.9940029985007486, 2.0389805097451275, 2.016491754122939, 2.01649175412294, 2.
 ↪0239880059970004, 1.9715142428785588, 1.7991004497751129, 1.6116941529235393, 1.
 ↪5142428785607198, 0]
vy : [0, 2.038980509745128, 2.0164917541229386, 2.0914542728635683, 2.0689655172413794,␣
 ↪1.9940029985007486, 2.0389805097451275, 2.016491754122939, 2.01649175412294, 2.
 ↪0239880059970004, 1.9715142428785588, 1.7991004497751129, 1.6116941529235393, 1.
 ↪5142428785607198, 0]
```

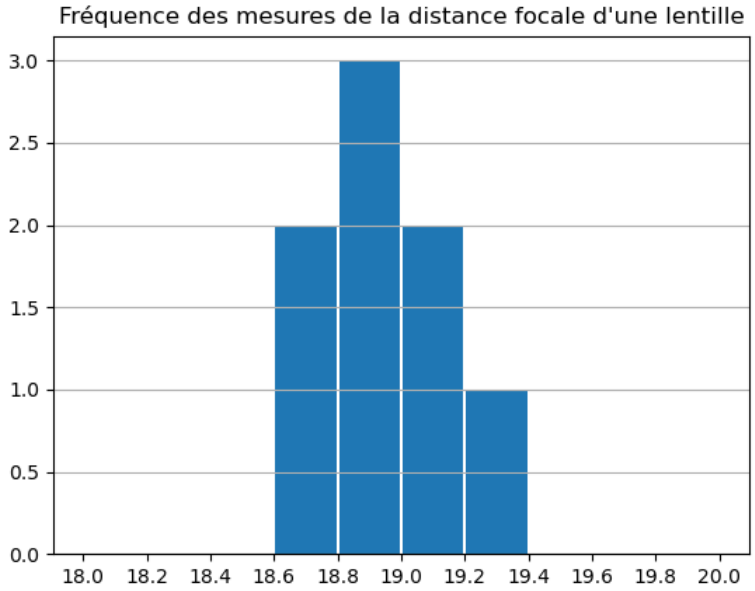
Exercice 14. (histogramme et incertitude-type)

```
import matplotlib.pyplot as plt
import statistics as stat

# répétabilité de la mesure d'un volume équivalent pour plusieurs groupes
f = [19.1, 18.9, 18.7, 19.0, 18.9, 19.2, 18.8, 18.7] # Listes des mesures

# HISTOGRAMME
freq, x, _ = plt.hist(f, range=(18, 20), bins=10, rwidth=0.95)
plt.title("Fréquence des mesures de la distance focale d'une lentille")
plt.grid(axis='y')
plt.xticks(x)
plt.show()

# CALCULS
N = len(f)
```

(suite sur la page suivante)

(suite de la page précédente)

```
f_moy = stat.mean(f)
s = stat.stdev(f)
ua_f = s/stat.sqrt(N)

# AFFICHAGE
print("f_prime = ", round(f_moy,2), "+/-", round(ua_f,2), "mm")
```




```
>>> %Run
f_prime =  18.91 +/- 0.06 mm
```

7.3 Bibliographie

— Cette documentation :

FIG. 1 – https://python.david-therincourt.fr/