Python pour la physique-chimie

David THERINCOURT
Lycée Roland Garros - Académie de la Réunion
8 avril 2021

Plus d'informations sur https://physique-chimie-python.readthedocs.io/

Table des matières

Mise à jour du 01/04/2021

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** (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 3.8 (2021) est la dernière version stable.

Avertissement : Il y a eu quelques changements notables au passage de Python 2 à Python 3, ce qui fait 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

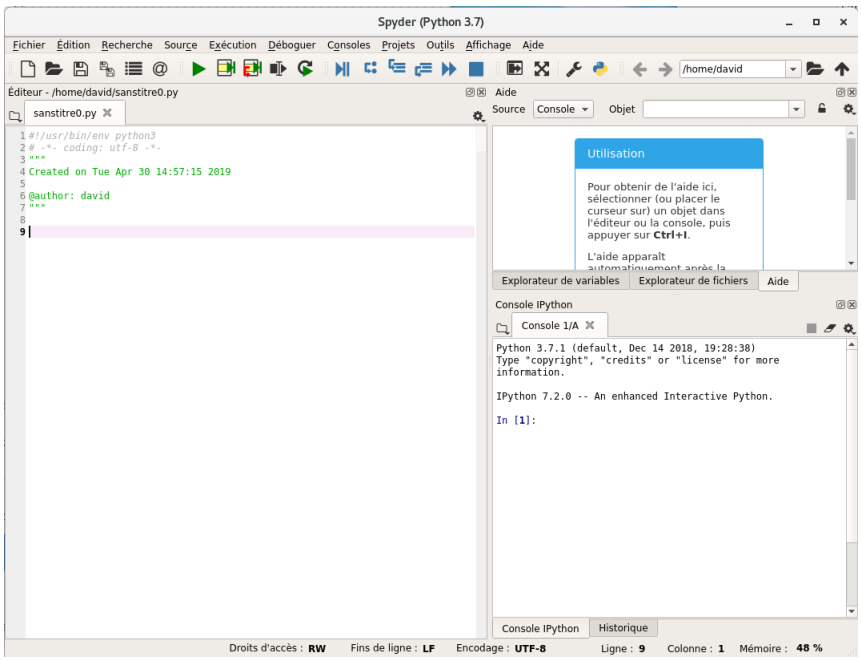
La distribution **Anaconda Python** est très utilisée par la communauté Python pour plusieurs raisons :

— 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'outil **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 sont écrites puis enregistrées 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 des avantages non-négligeables comme par exemples :

— EduPython peut s'installer ou s'utiliser à partir d'une clé USB ;
— la plupart des librairies utilisées au lycée sont déjà installées ;
— EduPython 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 Python en ligne

Il est également possible de programmer en Python dans un navigateur Web.

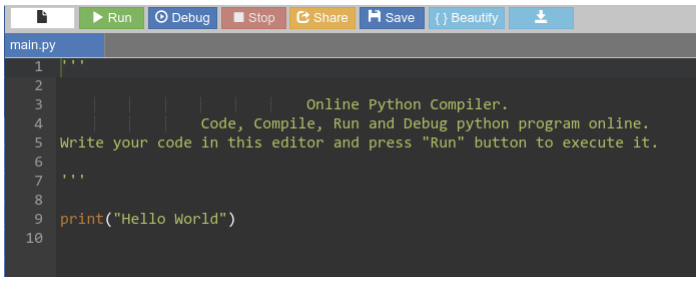


Fig. 4 – https://www.onlinegdb.com/online_python_compiler

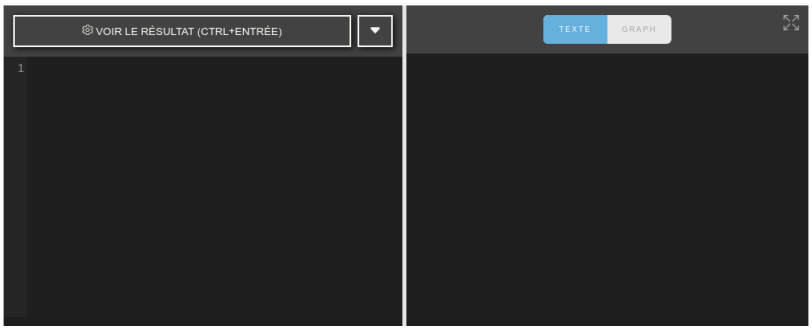


Fig. 5 – https://www.lelivrescolaire.fr/console-python

Avertissement : Attention, certaines fonctionnalités évoluées ne sont disponibles !

1.3 Premier pas avec Python

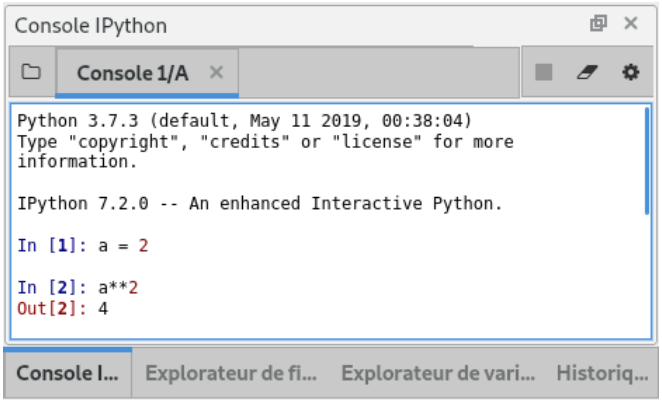
Voici une première instruction Python :

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

Cette instruction peut-être exécutée de deux façons.

1.3.1 Directement dans la console Python

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

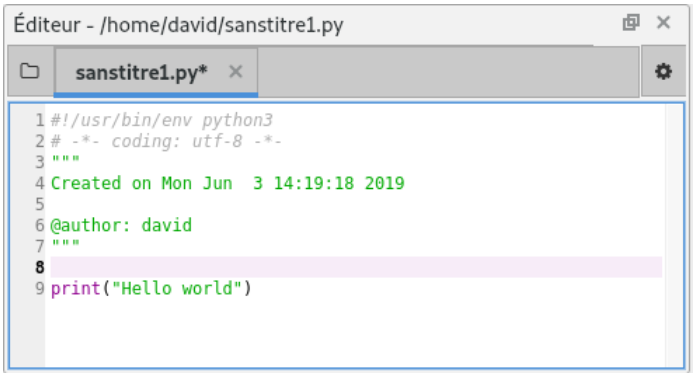


— Ici, la console (**interpréteur**) est IPython.
— `In [1]:` est une entrée numérotée de la console.
— `Out[1]:` est la sortie donnant le résultat de l'interprétation de l'entrée `In[1]:`.

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

1.3.2 A partir d'un script dans l'éditeur

Les instructions Python sont sauvegardées dans un fichier texte appelé **script** toujours avec l'extension `.py`.



— Les instructions Python sont écrites séquentiellement dans un éditeur de texte (ici l'éditeur de Spyder).
— Puis le script sera exécuté dans la console IPython à partir du menu `Exécution > Exécution`.

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 réalisée sous la **forme d'exemples commentés** dans une console Python 3 en mode interactif (à la manière d'une calculatrice).

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

2.1 Variables et types de base

2.1.1 Qu'est-ce qu'une variable ?

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

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

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

2.1.2 Affichage d'une variable

```
>>> 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.

2.1.3 Les types d'une variable

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

2.1.3.1 Les 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.

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

— Les **opérateurs mathématiques** s'appliquent normalement.
— L'opérateur `//` donne le **quotient** de la division entière.
— L'opérateur `%` (modulo) donne le **reste** de la division entière.

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

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

— En Python, la **taille d'un entier n'a pas de limite** !

2.1.3.2 Les flottants

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

```
>>> 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 le virgule.

2.1.3.3 Les booléens

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

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

— Un booléen prend les valeurs `True` (vrai) ou `False` (faux).

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

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

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

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

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

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

2.1.3.4 Les chaînes de caractères

Une chaîne de caractères est un **ensemble de caractères**.

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

— Le type `str` pour *string* (chaîne de caractères).
— Les chaînes de caractères sont toujours délimitées par les caractères `'` ou `"`.

```
>>> ch2 = "Paul"
>>> ch1 + ch2
'BonjourPaul'
```

(suite sur la page suivante)

(suite de la page précédente)

```
>>> 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 (`string`).

```
>>> 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 !
```

Par contre, la fonction `print()` permet l'affichage de tout type en texte.

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

2.1.4 Saisir le contenu d'une variable

En python, il est possible de demander 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.
— Le chaîne de caractère est affectée à la variable `rep`.

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

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

```
>>> 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.

2.2 Les types évolués

2.2.1 Les listes

En sciences physiques, l'utilisation des tableaux de données est monnaie courante. En langage Python, les **tableaux sont représentés par les listes**.

```
>>> l = [0, 'a', 4.13]
>>> print(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**.

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

(suite sur la page suivante)

(suite de la page précédente)

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

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

```
>>> l[1] = 3
>>> print(l)
[0, 3, 4.13]
```

— Les éléments d'une liste peuvent-être **modifiés**.

```
>>> 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.

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

— La fonction `len()` donne le nombre d'éléments dans une liste.

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

— La méthode `append()` ajoute un élément à la fin de la liste.
— La méthode `sort()` trie une liste dans l'ordre croissant.
— Application de ces deux méthodes modifient la liste sur laquelle elles sont appliquées !

2.2.2 Les tuples

Un tuple est un tableau **non-modifiable**.

```
>>> t = (1,2,3)
>>> t[1]
2
>>> t[1] = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

— Un tuple est une **série de valeurs** entre parenthèses `()` séparées par des virgules.
— Les éléments d'un tuple sont **non modifiables**.

```
>>> t = 1,2,3
>>> t
(1, 2, 3)
```

— Il est possible d'omettre les parenthèses (quand c'est possible) avec les tuples !

```
>>> a,b,c = 4,"azerty",4.56
>>> print(a)
4
>>> print(b)
azerty
>>> print(c)
4.56
```

— En Python, les **tuples permettent l'affectation multiple** de plusieurs variables sur une même ligne.

2.3 Les structures de contrôle

2.3.1 Les conditionnelles

Les structures conditionnelles permettent de faire des tests.

```
SI <condition>
ALORS
   <bloc instruction(s) pour condition vraie>
SINON
   <bloc instruction(s) pour condition fausse>
```

Avertissement : Dans les exemples qui suivent, les caractères `...` composent un **prompt secondaire** de la console Python signifiant que des instructions sont écrites sur plusieurs lignes. Ces caractères n'apparaissent jamais dans un script Python comme pour les trois chevrons >>> !

```
>>> if True :
...     print("Vrai")
... else :
...     print("Faux")
...
Vrai

>>> if False :
...     print("Vrai")
... else :
...     print("Faux")
...
Faux
```

— Une condition ne peut-être que vrai (`True`) ou fausse (`False`).
— Le caractère `:` signifie le **début d'un bloc d'instructions**.
— Un bloc d'instructions est toujours **indenté** (décalé du même nombre de caractères pour chaque ligne).
— Le mot clé `else` peut-être omis si rien ne doit-être fait pour une 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 !")
...
a plus petit ou égal à b !
>>> a>b
False
```

— Une condition s'obtient à partir d'une **expression** (exemple. `3>2`) dont le résultat est du type booléen (`True` ou `False`)

2.3.2 Les boucles

En programmation, une boucle permet de **répéter plusieurs fois ou indéfiniment des instructions**.

2.3.2.1 La boucle FOR

```
POUR <variable> DANS <liste> FAIRE
   <bloc instructions>
```

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

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

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

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

— La fonction `range(n)` facilite la création de boucle.
— La fonction `range(n)` renvoie un itérateur de `0` à `n-1`.
— La variable `i` de type entier est un **compteur**.

```
>>> for i in range(2,5):
...     print(i)
...
2
3
4
>>> for i in range(2,9,3):
...     print(i)
...
2
5
8
```

— Il existe d'autres formes de la fonction `range()`.

Application : Calcul d'une moyenne d'une liste de notes.

En parcourant la liste :

```
listeNote = [12, 15, 14, 16, 13, 15]
nbNote = len(listeNote)
somme = 0
for note in listeNote:
    somme = somme + note
moyenne = somme/nbNote
print(moyenne)
```

Avec un compteur :

```
listeNote = [12, 15, 14, 16, 13, 15]
nbNote = len(listeNote)
somme = 0
for i in range(nbNote):
    somme = somme + listeNote[i]
moyenne = somme/nbNote
print(moyenne)
```

Exemples :

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)

2.3.2.2 La boucle While

```
TANT QUE <condition> FAIRE
   <bloc instructions>
```

La boucle TANT QUE est utilisée quand le **nombre d'itérations n'est pas connu à l'avance**.

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

— La boucle s'effectue indéfiniment tant que la réponse est fausse !

```
from random import random
x=0
while x<10:
    x=x+3*random()
    print(x)
```

— La fonction `random()` renvoie un nombre (flottant) au hasard entre 0 et 1 (exclu).
— Ici le nombre d'itérations varie à chaque exécution du programme !

Exemples :

Évolution d'un système chimique (classe de première)

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

2.4 Les fonctions

2.4.1 Principe

En programmation, une fonction **réalise un traitement** puis **renvoie le résultat** de ce traitement.

```
>>> def aireCarre(a) :
...     aire = a*a
...     return aire
...
>>> aireCarre(5)
25
```

— Le mot clé `def` est toujours utilisé pour définir une fonction.
— Ici `aireCarre` est le **nom 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.
— Le nom de la fonction est utilisé lors de **l'appel** de la fonction avec des parenthèse pour préciser l'argument.

```
>>> def aireRectangle(l, L):
...     aire = l*L
...     return aire
...
>>> aireRectangle(4,5)
20
```

— Une fonction peut admettre aucun, un ou plusieurs arguments

2.4.2 Exemple 1 : calcul d'une énergie potentielle

Définition de la fonction calculant l'énergie potentielle de pesanteur dans un script Python :

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

Avertissement : Ne pas oublier d'exécuter le script Python pour que la fonction soit prise en compte !

Appel de la fonction dans l'interpréteur Python :

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

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

2.4.3 Exemple 2 : calcul d'une énergie mécanique

Définitions des fonctions :

```
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)
```

Résultats :

```
>>> Epp(50,10)
4905.0
>>> Ec(50,13)
4225.0
>>> Em(50,10,13)
9130.0
```

— La fonction `Em()` fait appel aux deux autres fonctions `Epp()` et `Ec()` !
— Pour chaque fonction, la variable `m` n'est pas la même. C'est toujours une variable locale.

2.5 Les modules

Le langage Python est fourni avec des fonctions de base. Mais elles sont insuffisantes dans certaines situations. Il est alors souvent utile d'utiliser des fonctions écrites par d'autres personnes de la communauté Python. Ces fonctions supplémentaires sont regroupées dans des librairies ou des modules.

2.5.1 Importation d'un module

Par exemple, il n'est pas possible d'effectuer une racine carrée à moins d'importer la fonction `sqrt()` du module `math` !

https://docs.python.org/3/library/math.html

```
>>> import math
>>> sqrt(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
>>> math.sqrt(2)
1.4142135623730951
```

— Le mot clé `import` charge la librairie `math` ;
— La fonction racine carrée n'est accessible qu'avec la syntaxe `math.sqrt()` signifiant que `sqrt()` est une fonction du module `math`.
— Toutes les autres fonctions mathématiques du module `math` sont ainsi disponibles de cette manière.

Si le nom du module est trop long à écrire à chaque fois, il est possible de faire comme dans l'exemple suivant :

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

— `mt` est un alias de `math`.

2.5.2 Importation d'une fonction particulière 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 module.

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

— La présence du caractère * importe toutes les fonctions du module `math` ;
— Par contre, c'est une mauvaise pratique **à éviter** tant que possible !

2.5.3 Modules pour les sciences physiques

Des modules sont souvent rencontrés en sciences physiques tels que `math`, `numpy`, `matplotlib`, `scipy`, ...

Par exemple :

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

Chapitre 3

SciPy

Scipy est une **collection de librairies** pour les sciences.

3.1 Les tableaux avec Numpy

Le module Numpy permet la création et la manipulation de tableaux (vecteurs) dans Python.

Site Web officiel de Numpy : https://www.numpy.org/

Documentation de Numpy : https://docs.scipy.org/doc/

3.1.1 Importation du module Numpy

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

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

3.1.2 Création de tableaux

3.1.2.1 A partir d'une liste

```
>>> 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.

3.1.2.2 A partir d'un intervalle et du nombre de d'éléments

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

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

3.1.2.3 A partir d'un intervalle et d'un pas

```
>>> 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`.

3.1.2.4 Créer un tableau vide

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

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

3.1.3 Manipulation de tableaux

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

— Les opérations mathématiques se font **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 !

```
>>> 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])
```

— Le module Numpy intégre ses propres fonctions mathématiques.

3.1.4 Importation et exportation de données

3.1.4.1 Fichier CSV

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

Le tableau de données suivant :

x	y	z
1	5	7
2	10	6
3	15	5
4	20	4

s'écrit comme ci-dessous dans un fichier texte au format CSV nommé par exemple `data.txt` :

```
x,y,z
1,5,7
2,10,6
3,15,5
4,20,4
```

— les données sont rangées en colonne ;
— les valeurs sont séparées par une virgule (ici), un point virgule ou une tabulation ;
— la première ligne renseigne sur les noms des variables.

3.1.4.2 Importer un fichier CSV

```
>>> import numpy as np
>>> np.loadtxt('data.txt', delimiter=',', skiprows=1, unpack=True)
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  10.,  15.,  20.],
       [ 7.,  6.,  5.,  4.]])
```

— La fonction `loadtxt()` importe les données d'un fichier CSV et renvoie un tableau Numpy.
— `delimiter=','` pour signifier que les virgules séparent les valeurs (le caractère spécial `\t` pour une tabulation).
— `skiprows=1` pour indiquer que la première ligne ne contient pas de données.
— L'option `unpack=True` transpose le tableau pour être dans le bon sens.

```
>>> import numpy as np
>>> x,y,z = np.loadtxt('data.txt',delimiter=',',skiprows=1,unpack=True)
>>> x
array([1., 2., 3., 4.])
>>> y
array([ 5., 10., 15., 20.])
>>> z
array([7., 6., 5., 4.])
```

— Une affectation multiple (utilisation d'un tuple) permet d'obtenir toutes les variables d'un coup.

3.1.4.3 Export dans un fichier CSV

```
import numpy as np
a = np.array( )                 # Données de la variable a
b = np.array([5, 6, 7, 8])      # Données de la variable b
data = np.transpose([a,b])      # Transposition des données
np.savetxt('data2.txt', data, delimiter=',', header='a,b', comments='')  # Création du fichier CSV
```

3.2 Les graphiques avec Matplotlib

Matplotlib est une librairie Python pour la visualisation de courbes.

Site Web officiel de Matplotlib : https://matplotlib.org/

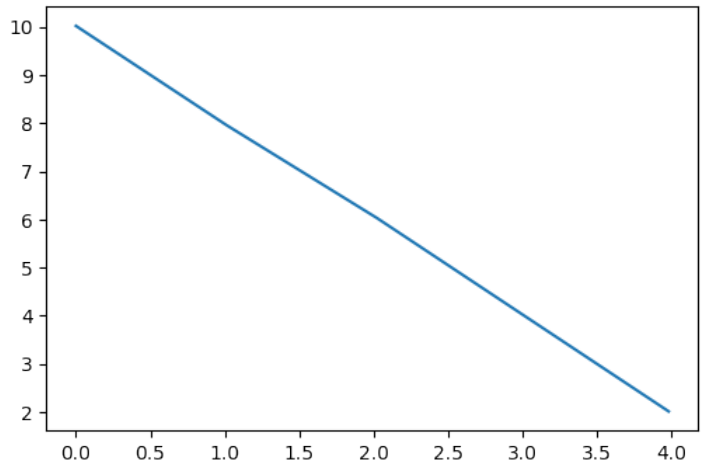
Référence de l'API de la collection *pyplot* de la librairie *matplotlib* :

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot

3.2.1 Tracer une courbe à partir de données

3.2.1.1 Les bases

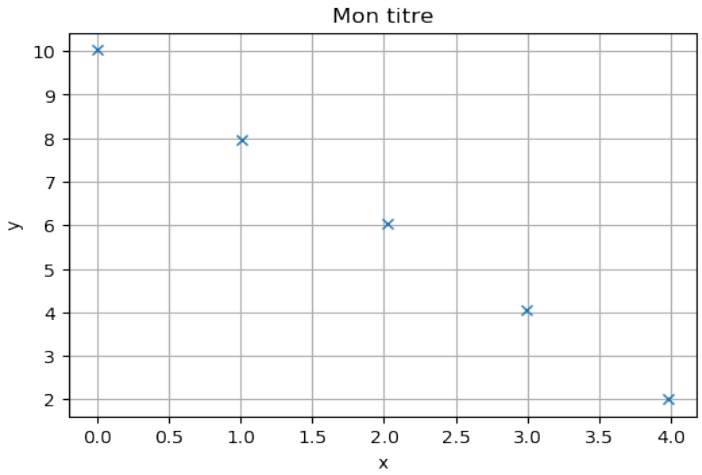
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([0, 1.01, 2.02, 2.99, 3.98])     # Données en abscisse
y = np.array([10.02, 7.96, 6.03, 4.04, 2.01]) # Données en ordonnée

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



— La collection `pyplot` du module `matplotlib` est importée avec l'alias `plt`.
— La fonction `plot()` trace la courbe `y=f(x)` à partir des tableaux `x` et `y`.
— La fonction `show()` appelée en dernier affiche la fenêtre graphique.

3.2.1.2 Ajouter un titre, une légende, une grille

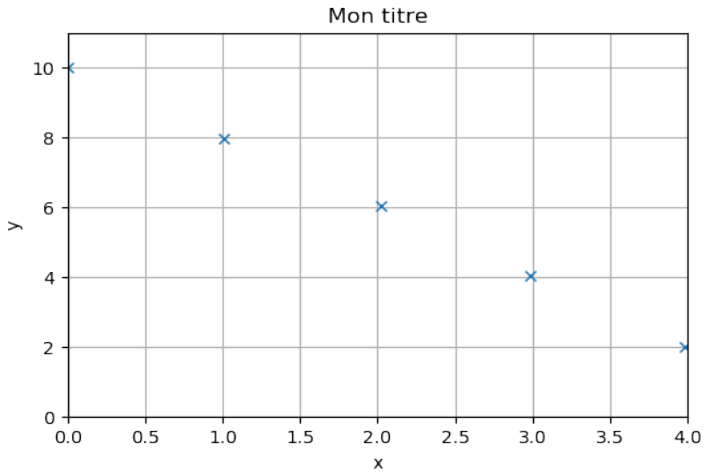
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([0, 1.01, 2.02, 2.99, 3.98])     # Données en abscisse
y = np.array([10.02, 7.96, 6.03, 4.04, 2.01]) # Données en ordonnée

plt.plot(x, y, 'x')     # Tracé de la courbe
plt.title('Mom titre')  # 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 `x` dans `plot()` met en évidence les points avec des croix 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 une grille.

3.2.1.3 Définir l'échelle

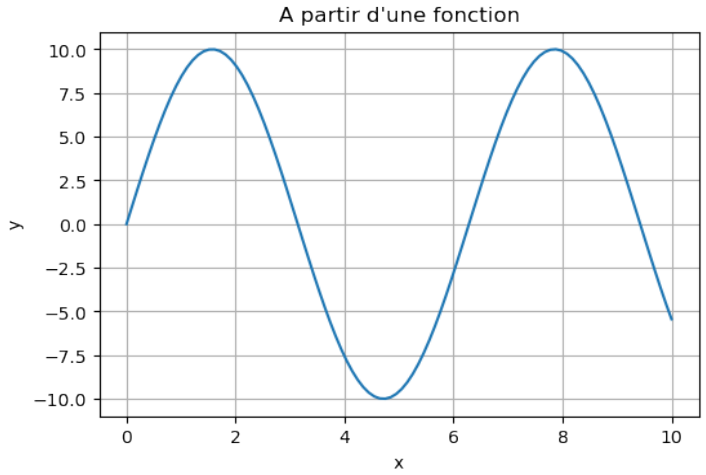
```
import numpy as np
import matplotlib.pyplot as plt


x = np.array([0, 1.01, 2.02, 2.99, 3.98])      # Données en abscisse
y = np.array([10.02, 7.96, 6.03, 4.04, 2.01]) # 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
```



3.2.2 Tracer une courbe à partir d'une fonction

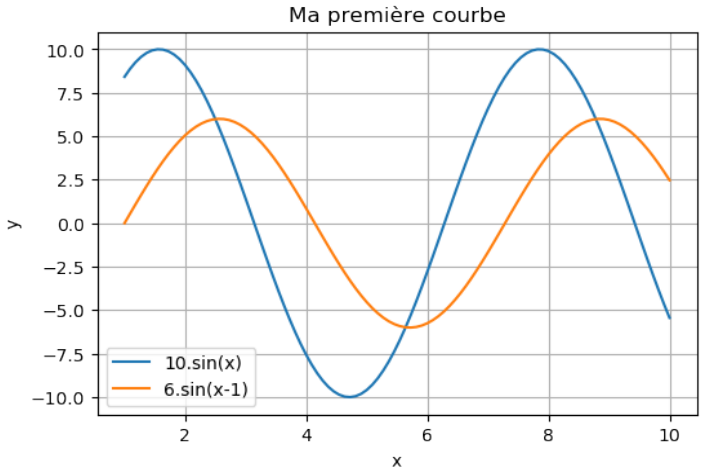
3.2.2.1 Cas d'une sinusoïde

```
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
```



3.2.2.2 Cas de deux sinusoïdes avec légende

```
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
```



— Dans la fonction `plot()`, le paramètre `label='...'` permet d'ajouter une étiquette dans la légende.

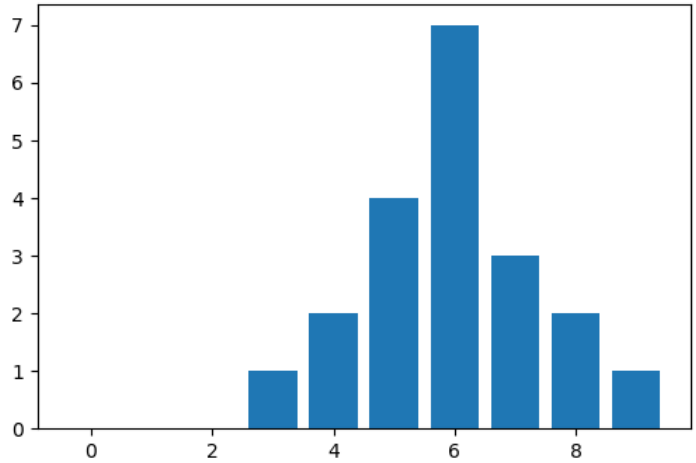
3.2.3 Tracer un histogramme

Le module **matplolib** propose la fonction `hist()` pour l'affichage d'un **histogramme** à partir d'un tableau de mesures. La documentation officielle de cette fonction se trouve ici.

3.2.3.1 Histogramme seul

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

x = [5, 6, 4, 7, 6, 7, 6, 8, 6, 5, 6, 5, 3, 9, 4, 6, 5, 8, 7, 6]

plt.hist(x, range=(0, 10), bins=10, rwidth = 0.8, align='left')
plt.show()
```

Résultats



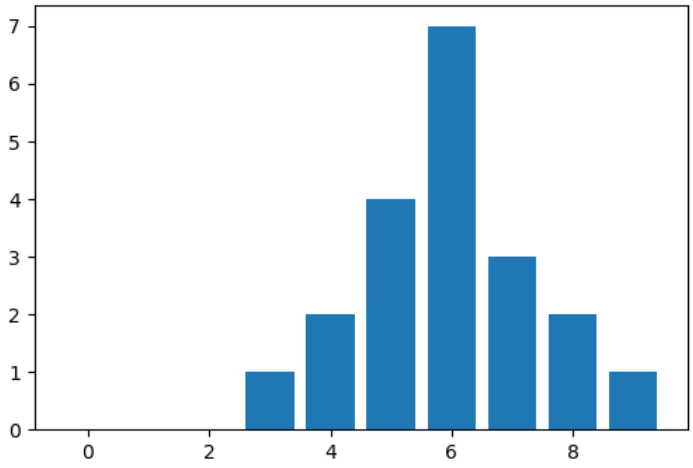
Dans la fonction `hist()` :

— `range=(0, 10)` fixe les limites de la plage d'étude du tableau de données.
— `bins=10` est le nombre d'intervalles dans la plage d'étude.
— `rwidth = 0.95` fixe la largeur des barres à 95% pour une meilleure visibilité.
— `align='left'` centre les barres.

3.2.3.2 Histogramme, valeurs et fréquences

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

x = [5, 6, 4, 7, 6, 7, 6, 8, 6, 5, 6, 5, 3, 9, 4, 6, 5, 8, 7, 6]

freq, valx, opt = plt.hist(x, range=(0,10),bins=10,rwidth = 0.8, align='left')
plt.show()

print("Valeur moyenne = ", np.mean(x))
print("Ecart type = ",np.std(x).round(2))
print('frequence = ',freq)
print('valeurs x = ',valx)
```

Résultats




```
Valeur moyenne =  5.95
Ecart type =  1.4309
frequence =  [0. 0. 0. 1. 2. 4. 7. 3. 2. 1.]
valeurs x =  [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

— Les fonctions de Numpy `mean()` et `std()` calculent respectivement la valeur moyenne et l'écart type.

Note : Le module `scipy.stats` fournit un grand nombre de lois de probabilités (Bernoulli, binomiale, normale, ...) et diverses méthodes de calcul (moyenne, médiane, variance, écart type, ...).

Voir la page Python pour le calcul scientifique/Statistiques sur WikiBooks

---

3.3 Le calcul scientifique avec Scipy

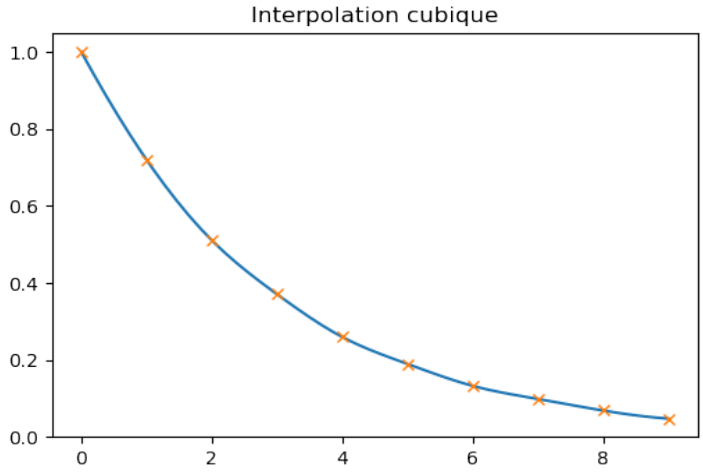
Site Web officiel de Scipy : https://www.scipy.org/

Documentation de Scipy : https://docs.scipy.org/doc/

3.3.1 Interpolation

La fonction `scipy.interpolate.interp1d()` retourne une fonction (mathématique) à une dimension interpolée à partir d'une série de points.

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

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

# Interpolation
f = interpolate.interp1d(x, y, kind='cubic')  # ou kind = 'linear'

xnew = np.linspace(0, 9, 50)        # Nouvelle abscisse pour f(x)
ynew = f(xnew)                      # Calcul des ordonnées de f(x)

plt.plot(xnew, ynew, '-')           # Tracé de la fonction
plt.plot(x, y, 'x')                 # Tracé des points
plt.show()                          # Affichage
```



3.3.2 Régression linéaire

La fonction `scipy.stats.linregress()` retourne les paramètres de la régression linéaire d'une série 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])
y = np.array([10.02, 7.96, 6.03, 4.04, 2.01])


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

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

plt.plot(xnew, ynew, '-')             # Tracé de la droite
plt.plot(x, y, 'x')                   # 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
```

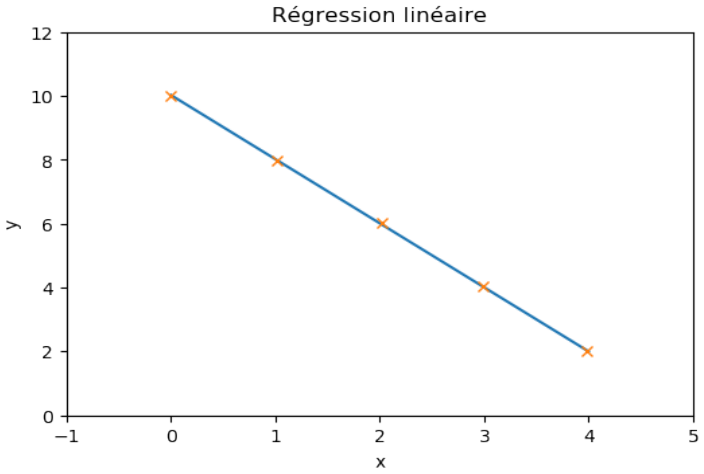
(suite sur la page suivante)

(suite de la page précédente)

```
plt.ylim(0, 12)                          # Echelle en ordonnée
plt.show()                               # Affichage
```

Résultats

```
a =  -2.0059103329622507
b =  10.023820665924502
rho =  -0.9999253712412117
```



— `rho` est le coefficient de corrélation linéaire.

3.3.3 Modélisation à partir d'un polynome

Cas d'une trajectoire parabolique de la forme :

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

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

# Donnée expérimentale
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]
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]


a, b, c = np.polyfit(X, Y, 2)  # Modélisation par un polynome de degré 2

print("a = ", a)               # Affichage
```

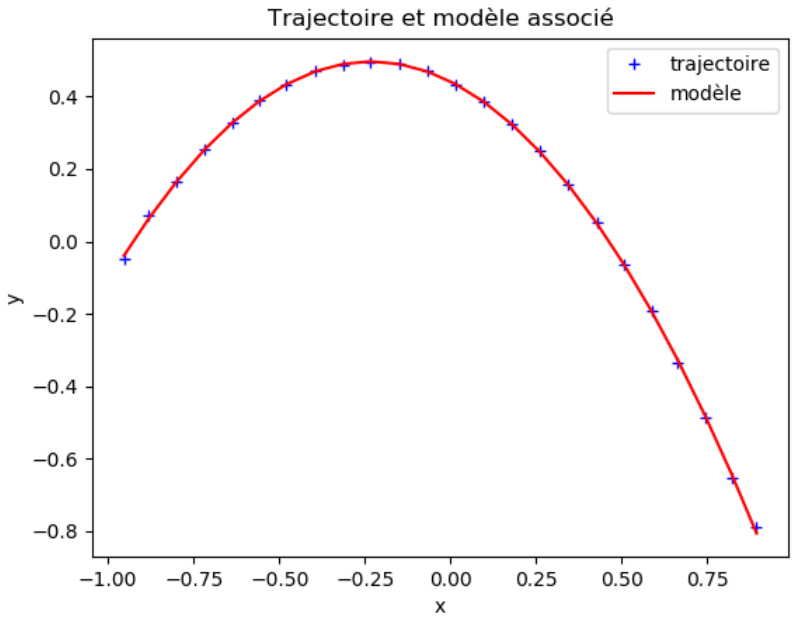
(suite sur la page suivante)

(suite de la page précédente)

```
print("b = ", b)
print("c = ", c)


X = np.array(X)               # Création d'un tableau Numpy pour X
Y_modele = a*X**2+b*X+c       # Création d'un tableau Numpy pour Y du modèle

plt.plot(X,Y,'b+', label = 'trajectoire')      # Courbe des mesures
plt.plot(X, Y_modele, '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 légend
plt.show()                                     # Affichage fenêtre
```

Résultats

```
a =  -1.028916427645116
b =  -0.47659343034449314
c =  0.4413962087861902
```



3.3.4 Modélisation à partir d'une fonction quelconque

La fonction `scipy.optimize.curve_fit()` retourne les paramètres de modélisation à partir d'une fonction quelconque.

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

x=np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y=np.array([0., 3.935, 6.321, 7.769, 8.647, 9.179, 9.502, 9.698, 9.817, 9.889])


def fct(x, A, tau):                    # Définition de la fonction
```

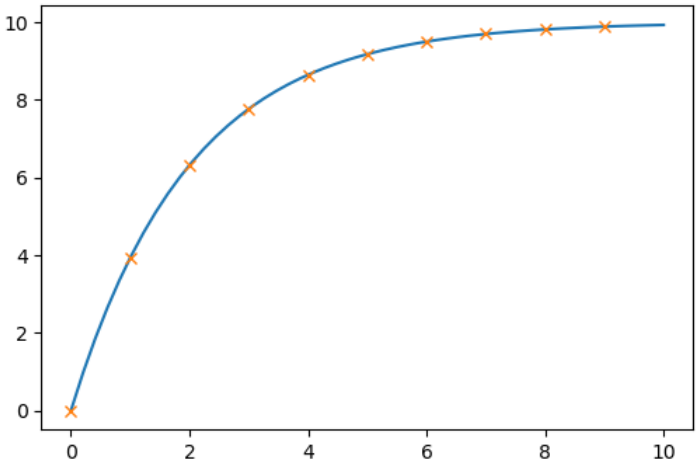
(suite sur la page suivante)

(suite de la page précédente)

```
    return A*(1-np.exp(-x/tau))         # Expression du modèle

(A, tau), pcov = curve_fit(fct, x, y)  # Détermination des paramètres du modèle
print("A = ", A)                       # Affichage de A
print("tau =", tau)                    # Affichage de tau

xnew = np.linspace(0, 10, 50)
ynew = fct(xnew, A, tau)
plt.plot(xnew, ynew, '-')
plt.plot(x, y, 'x')
plt.show()
```

Résultats

```
A =  9.99999510282223
tau = 1.9999259182304618
```



Chapitre 4

Nouveaux programmes du lycée 2019

4.1 Classe de seconde générale et technologique

4.1.1 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 ».

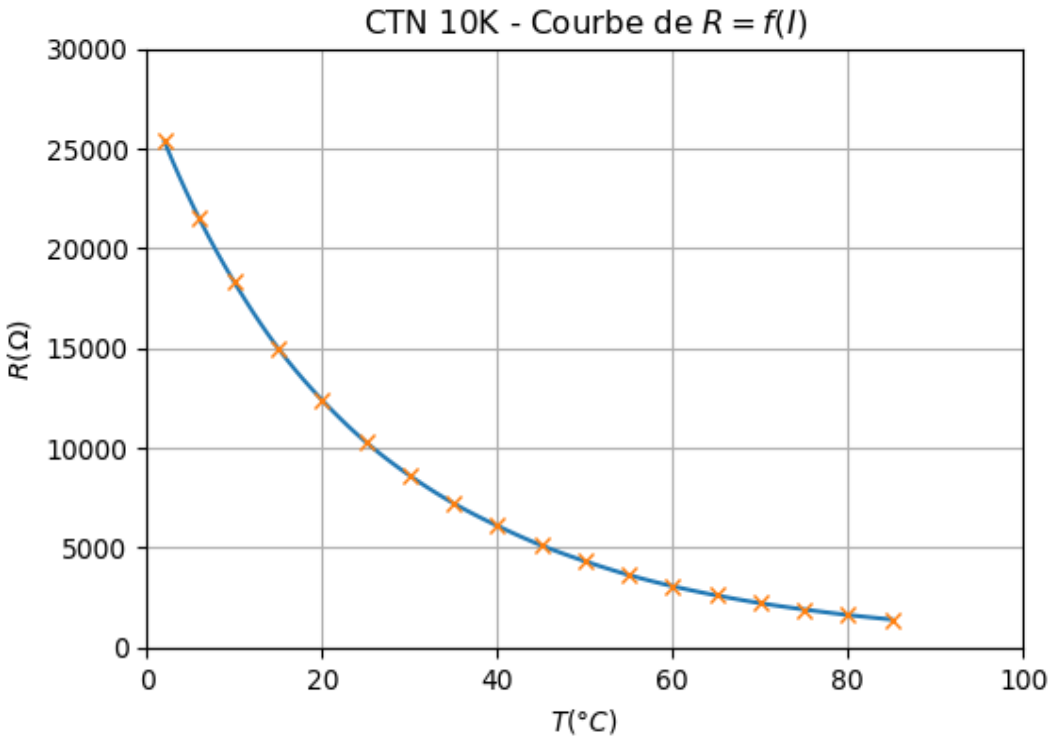
4.1.1.1 Caractéristique d'une CTN

Cas d'un capteur de température NTC 10K type EKS 221.

http://files.danfoss.com/technicalinfo/dila/01/RD8KC304_EKS221.pdf

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 100
from scipy import interpolate

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

#Interplotation
f = interpolate.interp1d(T, R, kind='cubic')
Tnew = np.linspace(T.min(), T.max(), 100)
Rnew = f(Tnew)

plt.plot(Tnew, Rnew)
plt.plot(T,R,'x')
plt.title('$R=f(T)$')
plt.xlabel('$T(°C)$')
plt.xlim(0,100)
plt.ylabel('$R(\Omega)$')
plt.ylim(0,30000)
plt.grid()
plt.show()
```

Résultats



4.1.2 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 ».

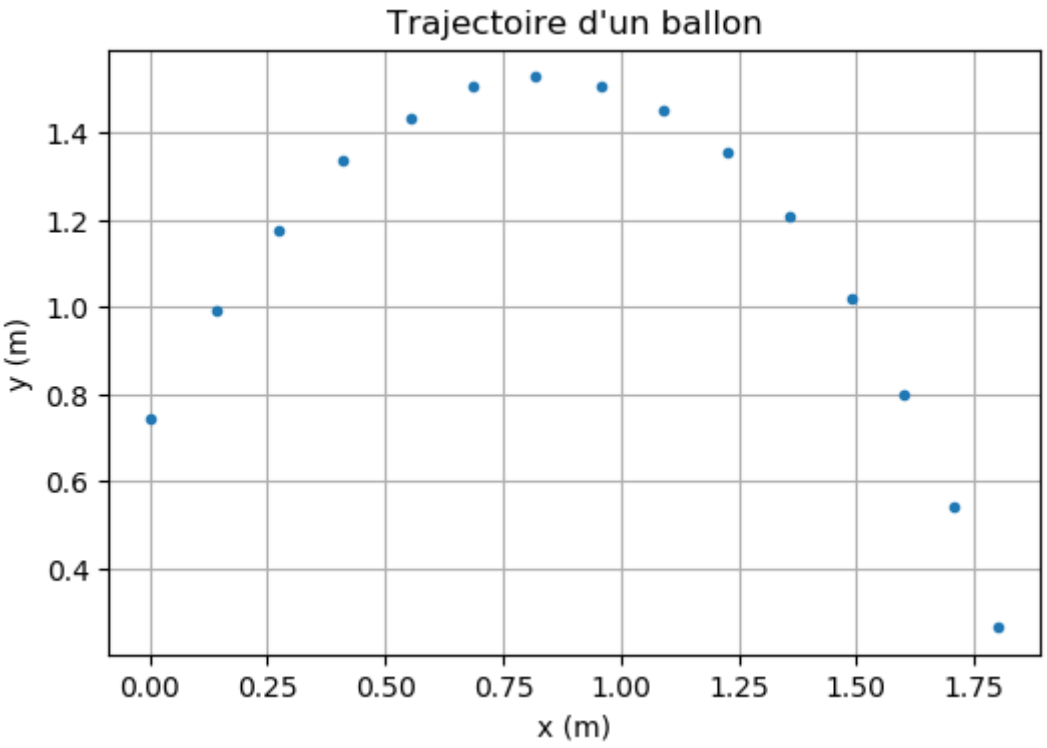
4.1.2.1 Principe

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

4.1.2.2 Programme Python

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

x = np.array([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 = np.array([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])

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

Résultats



4.1.3 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 ».

4.1.3.1 Principe

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

4.1.3.2 Programme Python

Méthode 1 La fonction `matplotlib.pyplot.arrow()` permet de tracer des flèches pour illustrer les graphiques.

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

# Données
dt = 0.0667
x = np.array([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 = np.array([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 vecteurs vitesses
N = len(x)          # Nombre de points de mesures
vx = np.zeros(N)    # Initialisation d'un tableau vide
vy = np.zeros(N)    # Idem
```

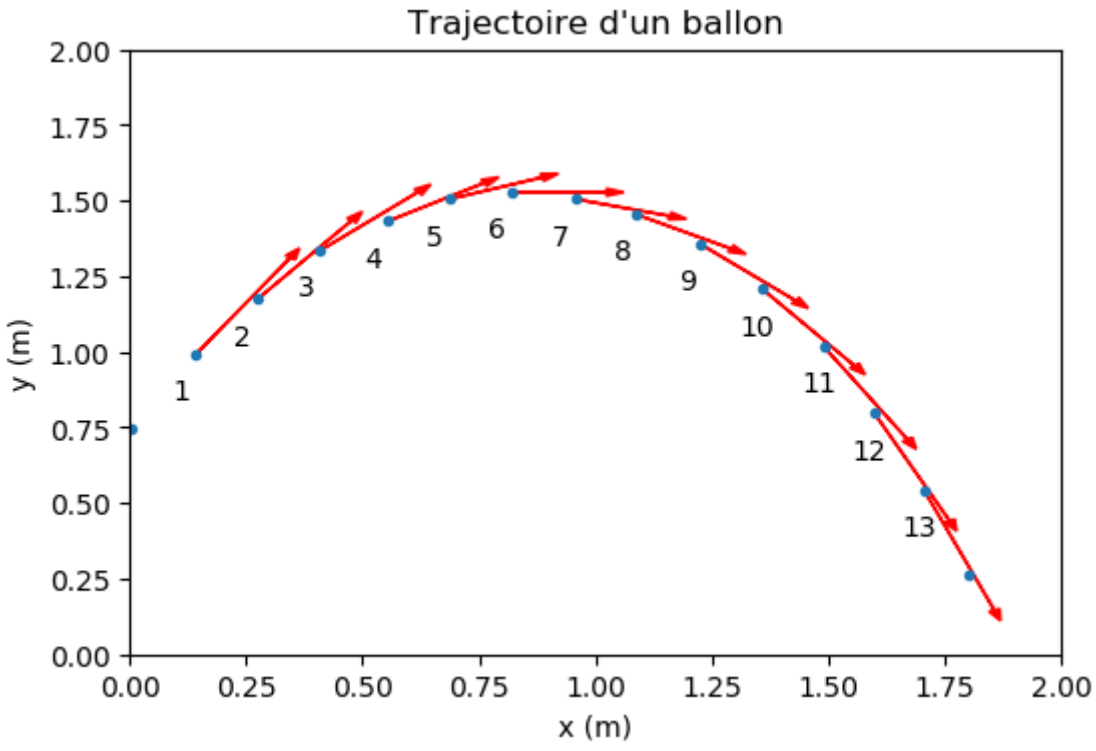
(suite sur la page suivante)

(suite de la page précédente)

```
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)

# Calcul des vitesses
v = np.sqrt(vx**2+vy**2)

plt.plot(x, y, '.')
plt.xlabel("x (m)")
plt.xlim(0, 2)
plt.ylabel("y (m)")
plt.ylim(0, 2)
plt.title("Trajectoire d'un ballon")
for i in range(1,N-1):
    plt.arrow(x[i], y[i], vx[i]/10, vy[i]/10, head_width=0.025, color='r')
    plt.annotate(i, (x[i]-0.05,y[i]-0.15))
    print('Point', i, '-> v=', round(v[i],2), " m/s")
plt.show()
```

Résultats

```
Point 1 -> v= 3.81  m/s
Point 2 -> v= 3.29  m/s
Point 3 -> v= 2.84  m/s
Point 4 -> v= 2.43  m/s
Point 5 -> v= 2.12  m/s
Point 6 -> v= 2.04  m/s
Point 7 -> v= 2.09  m/s
Point 8 -> v= 2.31  m/s
Point 9 -> v= 2.74  m/s
Point 10 -> v= 3.2  m/s
Point 11 -> v= 3.56  m/s
Point 12 -> v= 3.9  m/s
Point 13 -> v= 4.26  m/s
```

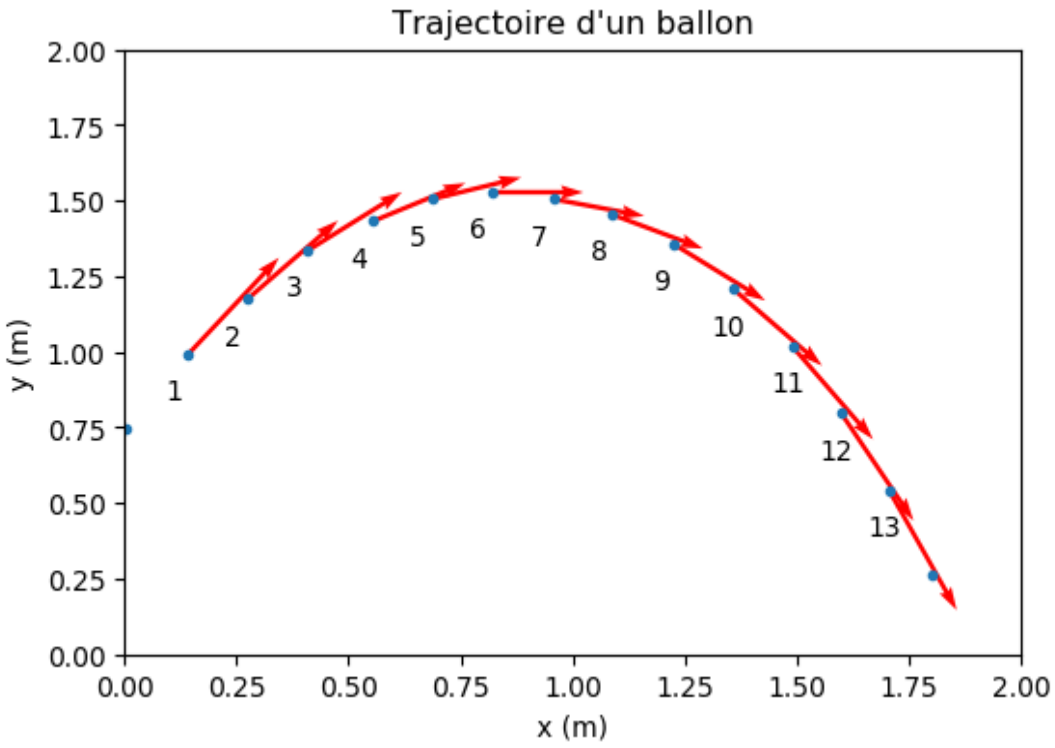


Méthode 2 La fonction `matplotlib.pyplot.quiver()` a l'avantage d'être itérative. Elle permet de tracer un champ de vecteur.

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

# Données
dt = 0.0667
x = np.array([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 = np.array([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 vecteurs vitesses
N = len(x)          # Nombre de points de mesures
vx = np.zeros(N)    # Initialisation d'un tableau vide
vy = np.zeros(N)    # Idem
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)

# Calcul des vitesses
v = np.sqrt(vx**2+vy**2)

plt.plot(x, y, ".")
plt.xlabel("x (m)")
plt.xlim(0, 2)
plt.ylabel("y (m)")
plt.ylim(0, 2)
plt.title("Trajectoire d'un ballon")
plt.quiver(x, y, vx, vy, angles='xy', scale_units='xy', scale=10, color='red', width=0.005)
for i in range(1, N-1):
    plt.annotate(i, (x[i]-0.05,y[i]-0.15))
    print('Point', i, '-> v=', round(v[i],2), " m/s")
plt.show()
```

Résultats

```
Point 1 -> v= 3.81  m/s
Point 2 -> v= 3.29  m/s
Point 3 -> v= 2.84  m/s
Point 4 -> v= 2.43  m/s
Point 5 -> v= 2.12  m/s
Point 6 -> v= 2.04  m/s
Point 7 -> v= 2.09  m/s
Point 8 -> v= 2.31  m/s
Point 9 -> v= 2.74  m/s
Point 10 -> v= 3.2  m/s
Point 11 -> v= 3.56  m/s
Point 12 -> v= 3.9  m/s
Point 13 -> v= 4.26  m/s
```



4.2 Classe première, enseignement de spécialité

4.2.1 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 ».

4.2.1.1 Programme Python

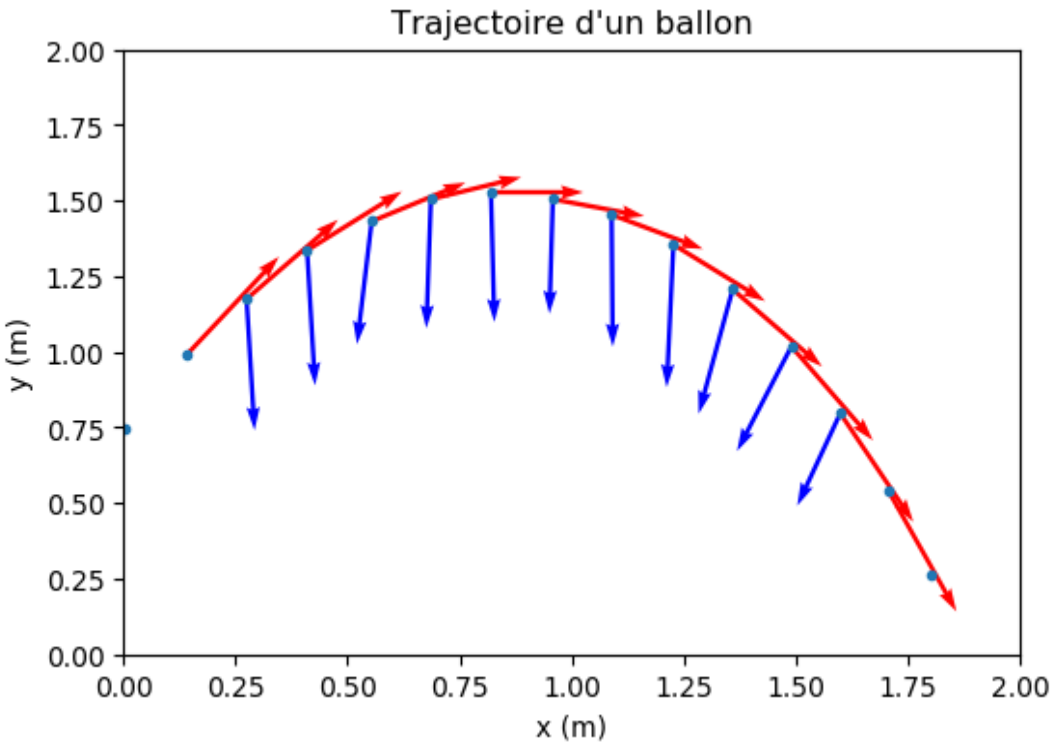
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 100

dt = 0.066
x = np.array([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 = np.array([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 = x.size

vx = np.zeros(N)
vy = np.zeros(N)
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)
```

(suite sur la page suivante)

(suite de la page précédente)

```
#print(vx.round(2))
#print(vy.round(2))

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

#print(dvx.round(2))
#print(dvy.round(2))

plt.plot(x, y, ".")
plt.quiver(x, y, vx, vy, angles='xy', scale_units='xy', scale=10, color='red', width=0.005)
plt.quiver(x, y, dvx, dvy, angles='xy', scale_units='xy', scale=3, color='blue', width=0.005)
plt.xlabel("x (m)")
plt.xlim(0, 2)
plt.ylabel("y (m)")
plt.ylim(0, 2)
plt.title("Trajectoire d'un ballon")
plt.show()
```

Résultats



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

4.2.2 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.

4.2.2.1 Exemple de réaction chimique

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 :

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

4.2.2.2 Etat final

```
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
    n_H2S = n_H2S - a*dx     # Nouvelle quantité de matière
    n_SO2 = n_SO2 - b*dx
    n_S   = n_S   + c*dx
    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))
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
```

4.2.2.3 Evolution des quantités de matière

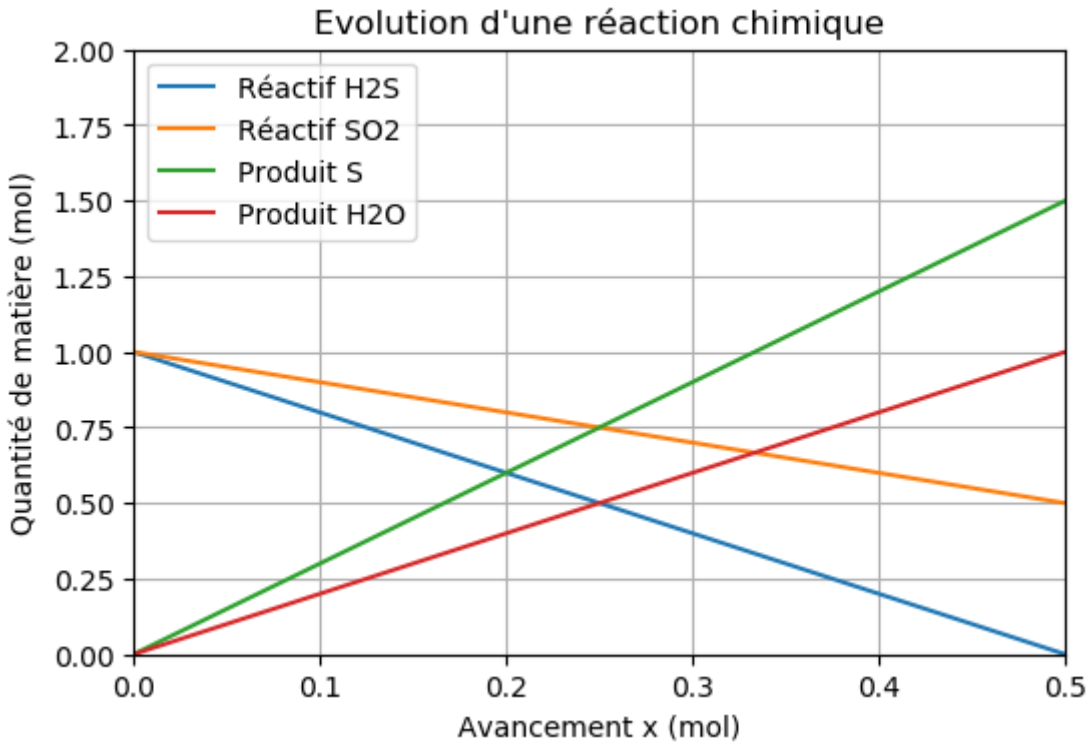
```
import matplotlib.pyplot as plt

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)
```

(suite sur la page suivante)

(suite de la page précédente)

```
    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



4.2.3 Bilan énergétique d'un mouvement rectiligne

Programme de première générale - Enseignement de spécialité - 2019

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

4.2.3.1 Principe

Étude d'un saut à l'élastique à partir d'une ressource EDUSCOL.

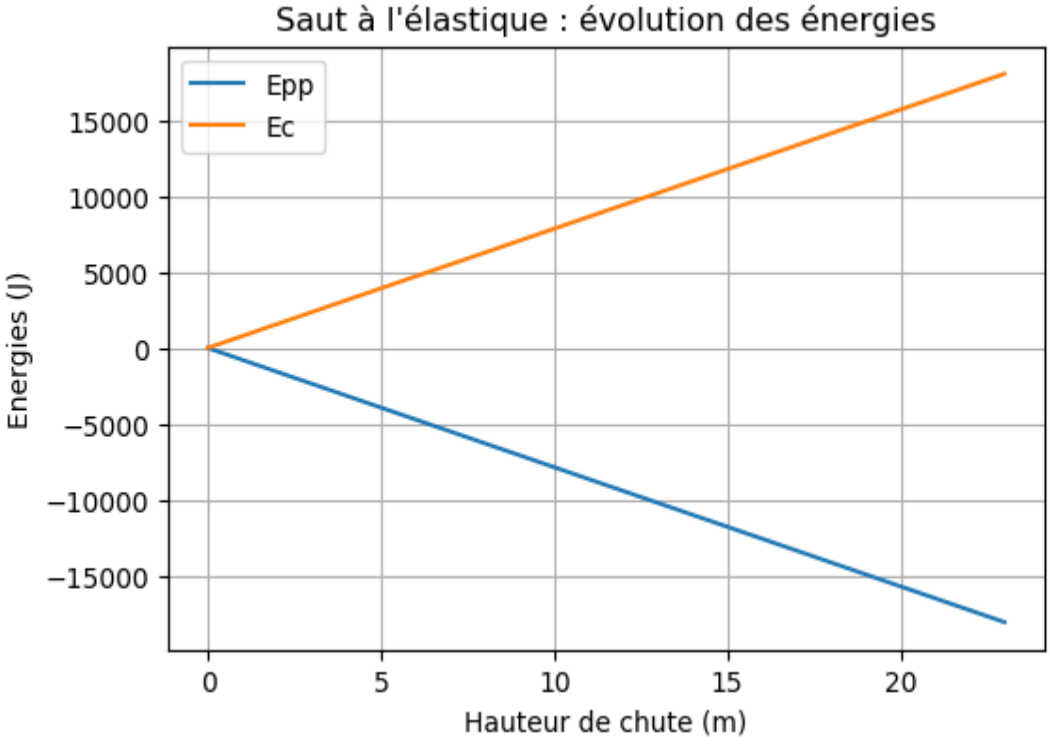
http://cache.media.education.gouv.fr/file/Programmer_en_physique-chimie/12/5/RA18_Lycee_PHCH_etude-energetique-mouvement-rectiligne_1044125.pdf

4.2.3.2 Avant que l'élastique soit tendu

Script Python

```
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 100

m, g, h0 = 80, 9.81, 0

Epp=[-m*g*h0]
Ec=[m*g*h0]
H=[h0]

for h in range(0, 24):
    H.append(h)
    Epp.append(-m*g*h)
    Ec.append(m*g*h)

plt.plot(H, Epp, label="Epp")
plt.plot(H, Ec, label="Ec")
plt.xlabel("Hauteur de chute (m)")
plt.ylabel("Energies (J)")
plt.title("Saut à l'élastique : évolution des énergies")
plt.legend()
plt.grid()
plt.show()
```

Résultats

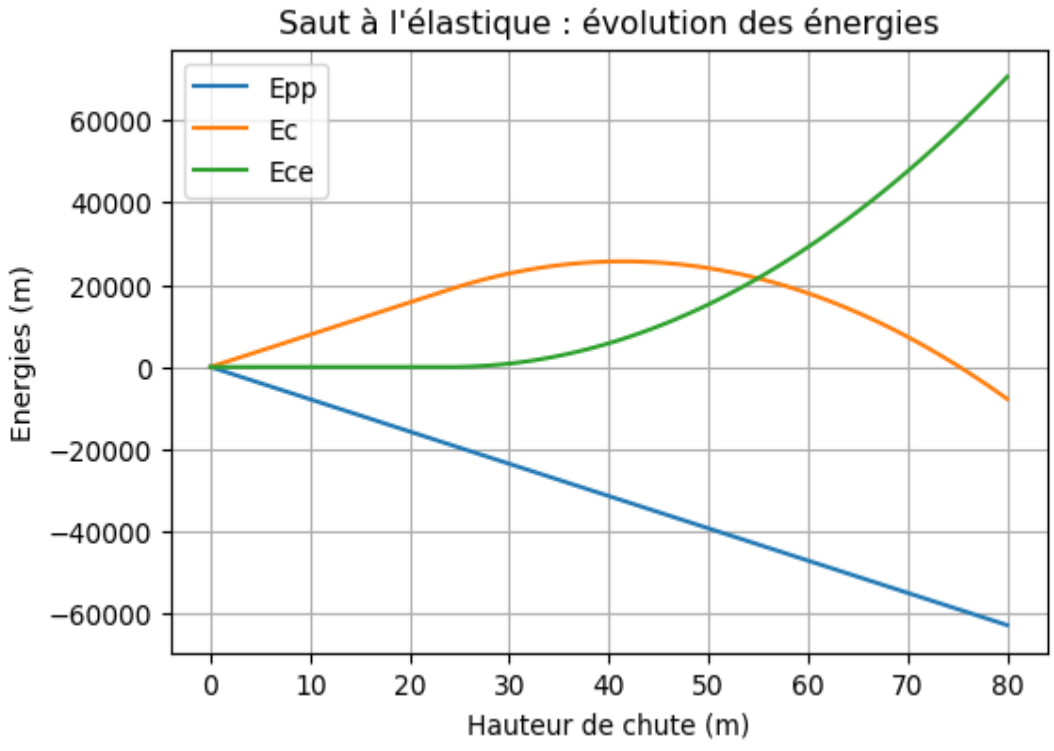


4.2.3.3 Chute complète

Script Python

```
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 100

m, g, h0, l0, k = 80, 9.81, 0, 24, 45

Epp=[-m*g*h0]
Ec=[m*g*h0]
Epe=[0]
H=[h0]

for h in range(1,81):
    H.append(h)
    Epp.append(-m*g*h)
    if h<l0:
        Epe.append(0)
        Ec.append(m*g*h)
    else:
        Epe.append(0.5*k*(h-l0)**2)
        Ec.append(m*g*h-0.5*k*(h-l0)**2)

plt.plot(H, Epp, label="Epp")
plt.plot(H, Ec, label="Ec")
plt.plot(H, Epe, label="Ece")
plt.legend()
plt.title("Saut à l'élastique : évolution des énergies")
plt.xlabel("Hauteur de chute (m)")
plt.ylabel("Energies (m)")
plt.grid()
plt.show()
```

Résultats



4.2.3.4 Chute complète avec calcul de longueur maximale

Script Python

```
A faire !
```

Résultats

4.2.4 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 ».

4.2.4.1 Programme Python

```
A faire !
```

Résultats

4.2.5 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 ».

4.2.5.1 Programme Python

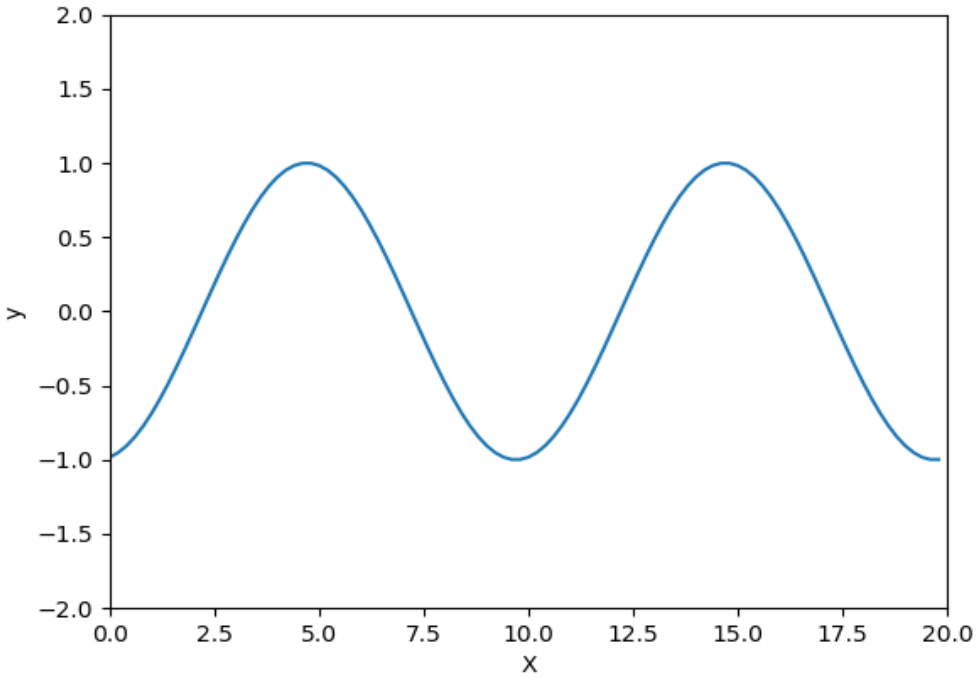
```
from math import sin, pi
import matplotlib.pyplot as plt
from matplotlib import animation

Xmax = 20.0
Tmax = 10.0
Ne = 100

positionsX = [i*Xmax/Ne for i in range(Ne)]
temps = [i*Tmax/Ne for i in range(Ne)]

T = 5
v = 2

mescourbes = [[sin(2*pi/T*(t-x/v)) for x in positionsX ] for t in temps]

fig = plt.figure()
ax = plt.axes(xlim=(0, Xmax), ylim=(-2, 2))
plt.xlabel("X")
plt.ylabel("y")
courbe, = ax.plot(positionsX, mescourbes[0])
```

(suite sur la page suivante)

(suite de la page précédente)

```
def incrementeTemps(i):
    courbe.set_ydata(mescourbes[i])
    return courbe

line_ani = animation.FuncAnimation(fig, incrementeTemps, 100, interval=100, blit=False)
plt.show()
```

Résultats



4.3 Classe terminale, enseignement de spécialité

4.3.1 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.

4.3.1.1 Exemple : mesures de la célérité d'un son

A l'aide d'un émetteur-récepteur ultrasons du type HC-SR04, un microcontrôleur Arduino effectue plusieurs fois la mesure de la célérité du son dans l'air.

Expérimentation Le programme Arduino utilisé pour obtenir, dans le moniteur série du logiciel Arduino, les mesures au format CSV est donné ci-dessous.

```
/*
 * Mesure de la célérité du son au format CSV
 * avec un microntrôleur EducaduinoLab
 */

#define pinTrig 19      // Module ultrason sur
#define pinEcho 18      // les broches D18/D19

float distance = 1.735; // Distance en module et réflecteur
long  dureeEcho;        // Durée mesurée
float celerite;         // célérité calculée
int n=1;                // Initialisation du compteur

void setup() {
  pinMode(pinTrig,OUTPUT);      // Broche Trig en sortie
  digitalWrite(pinEcho,LOW);    // Sortie Trig à l état bas
  pinMode(pinEcho,INPUT);       // Broche Echo en entrée
  Serial.begin(9600);           // Paramétrage du port série
  Serial.println("n;duree;v");  // Ecriture première ligne du CSV
}

void loop() {
  if (n<=100) {
  digitalWrite(pinTrig,HIGH);            // Déclenchement d'une mesure
  delayMicroseconds(10);                 // Attendre 10 microseconde
  digitalWrite(pinTrig,LOW);             // Fin impulsion (Etat bas)
  dureeEcho = pulseIn(pinEcho,HIGH);     // Mesure de la durée de l impulsion sur Echo
  celerite = 2*distance/dureeEcho*1E6;   // Calcul de la célérité
  Serial.print(n);                       // Début écriture ligne CSV
  Serial.print(";");
  Serial.print(dureeEcho);
  Serial.print(";");
  Serial.println(celerite);              // Fin écriture ligne CSV
  delay(100);                            // Attendre 100 ms
  n++;                                   // Incrémentation du compteur
  }
}
```

Le fichier CSV obtenu pour 100 mesures est téléchargeable ici `data_son.txt`.

Voici un extrait du fichier CSV des 30 premières mesures :

```
n;duree;v
1;9853;352.18
2;9934;349.31
3;9901;350.47
4;9933;349.34
5;9902;350.43
6;9901;350.47
7;9928;349.52
8;9928;349.52
9;9955;348.57
10;9928;349.52
11;9934;349.31
12;9928;349.52
13;9928;349.52
14;9933;349.34
15;9927;349.55
16;9980;347.70
17;9928;349.52
18;9929;349.48
19;9934;349.31
```

(suite sur la page suivante)

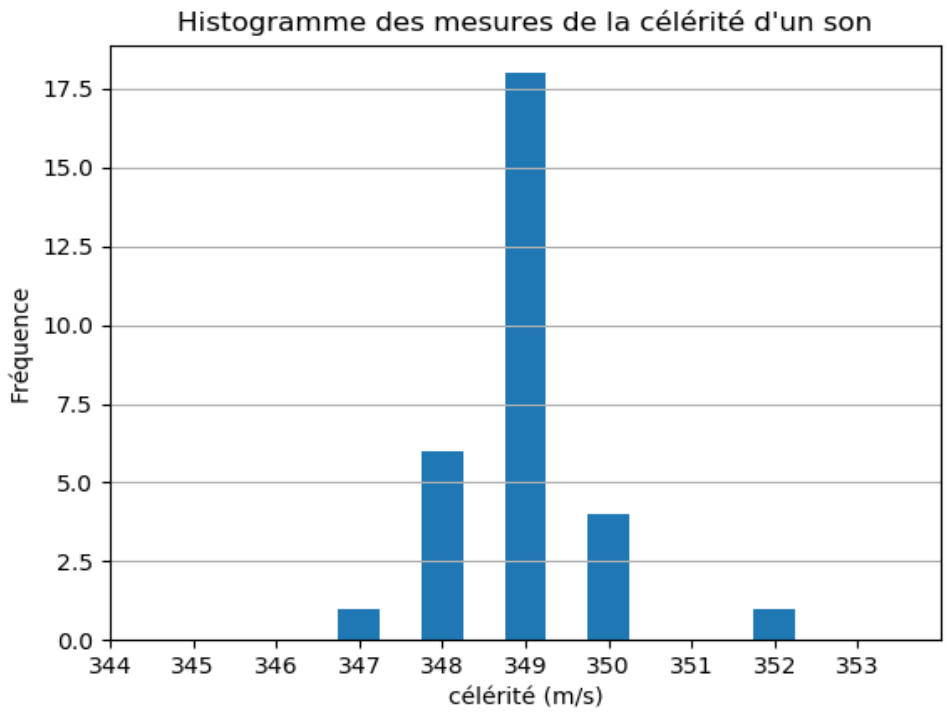
(suite de la page précédente)

```
20;9954;348.60
21;9901;350.47
22;9928;349.52
23;9954;348.60
24;9934;349.31
25;9955;348.57
26;9928;349.52
27;9934;349.31
28;9929;349.48
29;9956;348.53
30;9954;348.60
```

Avertissement : Le programme Python et le fichier CSV sont à **enregistrer dans le même répertoire de travail** !

Programme Python

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

# Importation des données au format CSV
n, duree, v = np.loadtxt("data_son.txt", delimiter=';', skiprows=2, unpack=True)

# Calcul des fréquences et affichage de l'histogramme
plt.hist(v, range=(344,354), bins=10, align='left', rwidth=0.5)

# Paramétrage de l'affichage
plt.title("Histogramme des mesures de la célérité d'un son")
plt.xlabel("célérité (m/s)")
plt.xlim(344, 354)
plt.xticks(np.arange(344, 354, 1))
plt.ylabel('Fréquence')
plt.grid(axis='y')
plt.savefig("ultrason_histogramme.png")
plt.show()

# Calcul
moy = np.mean(v)                          # valeur moyenne de l'échantillon
print("Moyenne =", moy)
ecart_type = np.std(v)                    # Ecart-type de l'échantillon
print("Ecart type =", ecart_type)
```




```
Moyenne = 349.4363333333333
Ecart type = 0.8026476326646889
```

4.3.2 Incertitudes-types compsées - 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.

4.3.3 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.

4.3.3.1 Dosage par titrage par suivi pH-métrie

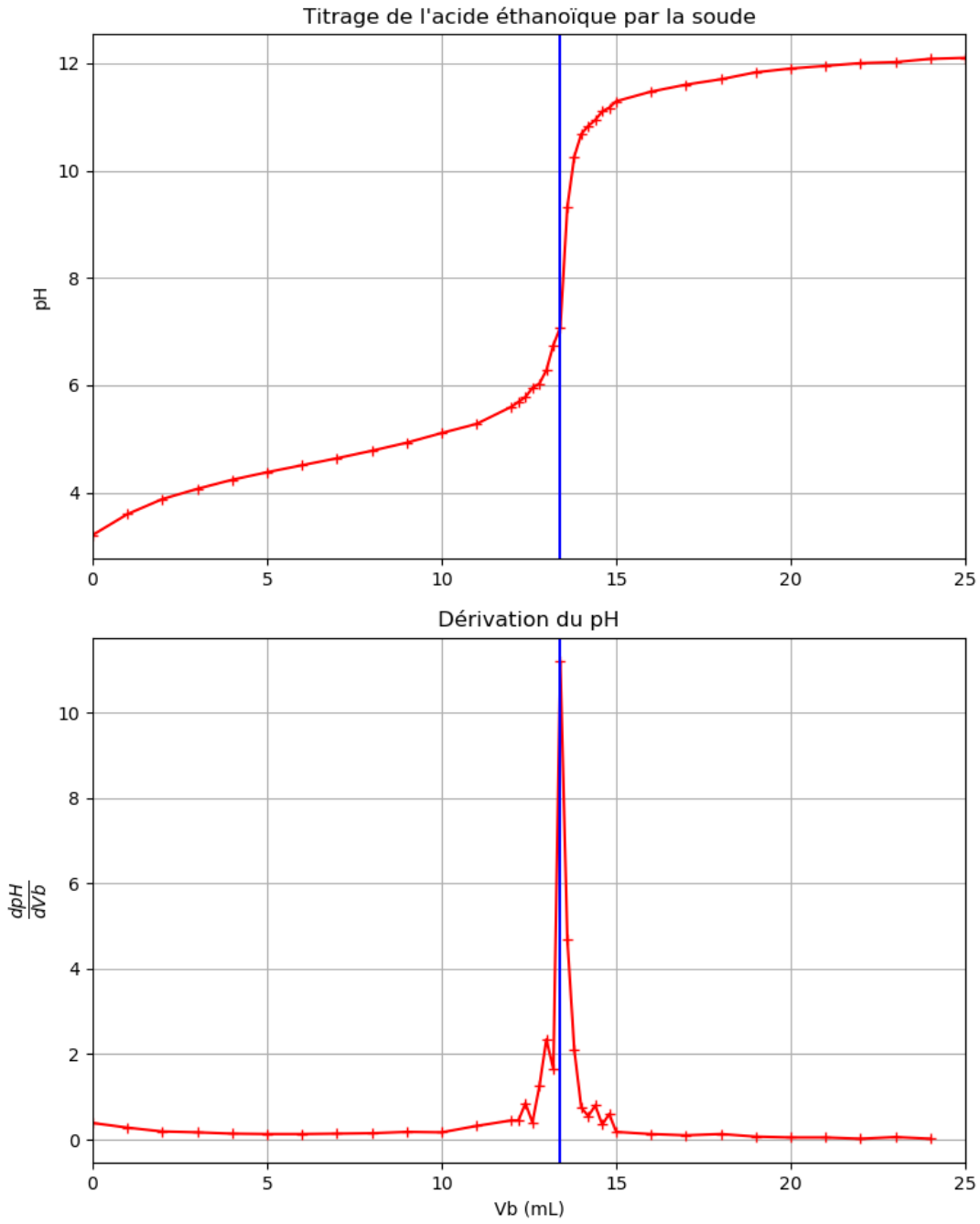
Programme Python

```
""" Titrage d'une solution aqueuse d'acide éthanoïque
par une solution aqueuse d'hydroxyde de sodium
"""

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

def derivee(x, y):
   y_prim = []
   for i in range (len(x)-1):
      k = (y[i+1]-y[i])/(x[i+1]-x[i])
      y_prim.append(k)
   return x[:-1], y_prim

plt.rcParams["figure.figsize"] = (8,10)  # width, height in inches (100 dpi)
plt.subplots_adjust(hspace=0.15, left=0.10, bottom=0.06, right=0.95, top=0.94 ) # Position en %

# Mesures
Vb = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,12.2,12.4,12.6,12.8,13,13.2,13.4,13.6,13.8,14,14.2,14.
↪4,14.6,14.8,15,16,17,18,19,20,21,22,23,24,25])
pH = np.array([3.21,3.60,3.88,4.07,4.24,4.38,4.51,4.64,4.78,4.93,5.11,5.28,5.60,5.69,5.78,5.95,6.
↪03,6.28,6.75,7.08,9.32,10.26,10.68,10.83,10.94,11.1,11.17,11.29,11.47,11.60,11.70,11.83,11.90,
↪11.95,12.00,12.02,12.08,12.10])

# Cacluls
Vb_new, derivee_pH = derivee(Vb, pH)
max = max(derivee_pH)
print("Max dérivée pH =", max)
Vbe = Vb_new[derivee_pH == max]
print("Volume à l'équivalence =", Vbe[0], "mL")

# Première courbe
plt.subplot(211)
plt.plot(Vb, pH, "r+-")
plt.axvline(Vbe, color="blue")
plt.title("Titrage de l'acide éthanoïque par la soude")
#plt.xlabel("Vb (mL)")
plt.xlim(0,25)
plt.ylabel("pH")
plt.grid()

# Seconde courbe
plt.subplot(212)
plt.plot(Vb_new, derivee_pH, "r+-")
plt.axvline(Vbe, color="blue")
plt.title("Dérivation du pH")
plt.xlabel("Vb (mL)")
```

(suite sur la page suivante)

(suite de la page précédente)

```
plt.xlim(0,25)
plt.ylabel("$\dfrac{dpH}{dVb}$")
plt.grid()

plt.show()
```



4.3.3.2 Dosage par titrage par suivi conductimétrie

Programme Python

```
"""
Simulation d'une courbe de titrage par conductimétrie
```

(suite sur la page suivante)

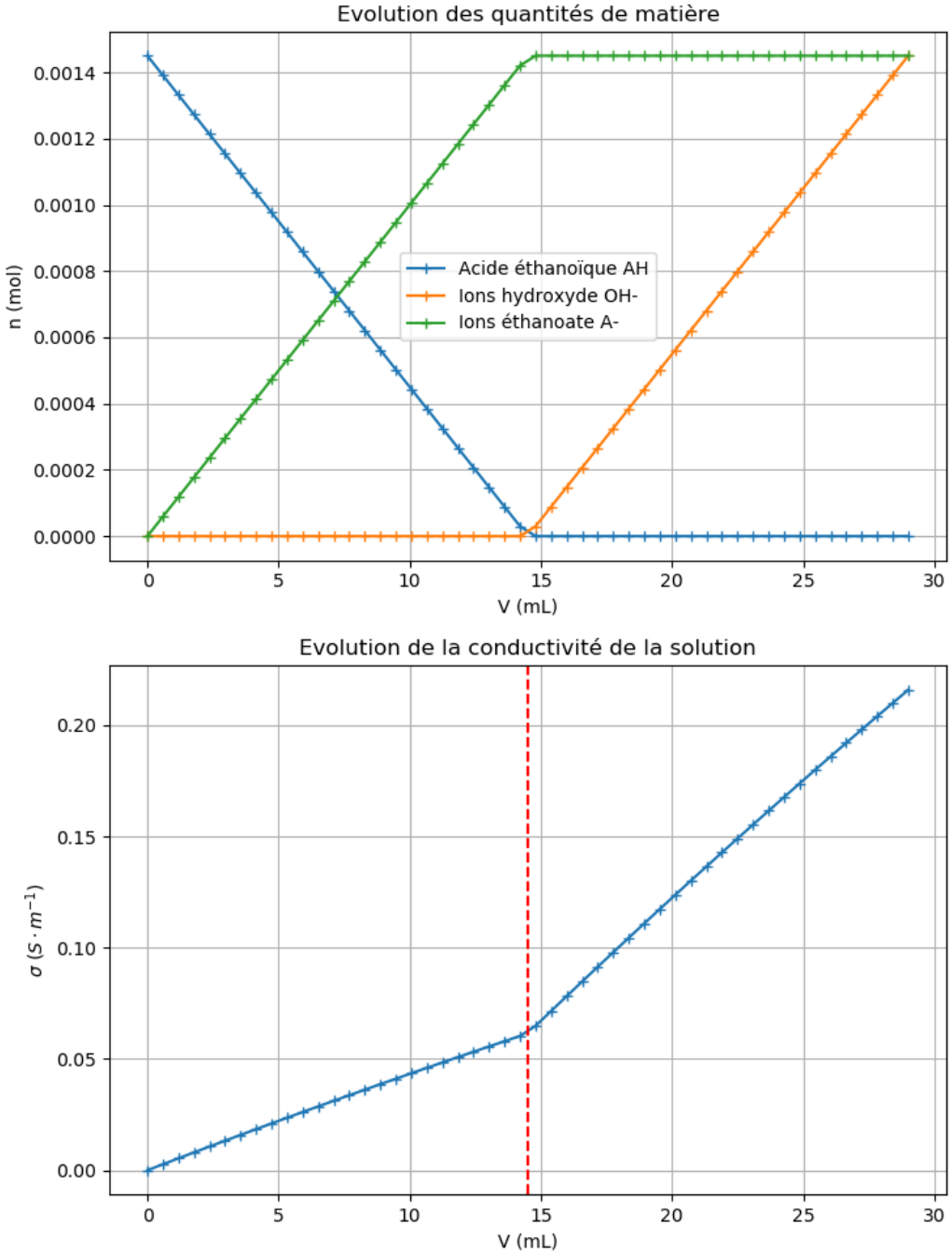
(suite de la page précédente)

```
AH(aq) + HO-(aq) -> A-(aq) + H2O(l)
Conductivité molaire ionique en mS.m2.mol-1 à 25 °C
lambda(Na+) = 5,0
lambda(HO-) = 20,0
lambda(A-)  = 4,1
"""

import numpy as np
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (8,10)  # width, height in inches (100 dpi)
plt.subplots_adjust(hspace=0.2, left=0.15, bottom=0.06, right=0.95, top=0.95 ) # Position en %

lambda_HO, lambda_Na, lambda_A = 20.0, 5.0, 4.1

Cb = 0.100         # Concentration base en mol/L
C_AH = 0.145       # Concentration acide en mol/L
Vo = 10.0E-3       # Volume solution d'acide éthanoïque
Veau = 200.0E-3      # Volume d'eau ajouté en L

Ve = C_AH*Vo/Cb    # Volume équivalent en mL calculé Ve = C_AH*Vo/Cb
NB = 50            # Nb points

V = np.linspace(0, 2*Ve, NB)                 # Initialisatin du tableau de volume

# Initialisation des tableaux de concentration
n_AH = np.zeros(NB)
n_HO = np.zeros(NB)
n_NA = np.zeros(NB)
n_A = np.zeros(NB)

for i in range(NB) :
   Vol = V[i]
   if Vol <= Ve :
      # Reactifs
      n_AH[i] = C_AH*Vo - Cb*Vol
      n_HO[i] = 0
      # Produits
      n_A[i] = Cb*Vol
      n_NA[i] = Cb*Vol
   else :
      print(i, Vol)
      # Reactifs
      n_AH[i] = 0
      n_HO[i] = Cb*(Vol-Ve)
      # Produits
      n_A[i] = Cb*Ve
      n_NA[i] = Cb*Vol

# Calcul du tableau de conductivité
sigma = (lambda_HO*n_HO + lambda_A*n_A + lambda_Na*n_NA) / (Veau + V)

# Courbe d'évolution des quantités de matière
plt.subplot(211)
plt.plot(V*1000, n_AH, "-+", label ='Acide éthanoïque AH')
plt.plot(V*1000, n_HO, "-+", label ='Ions hydroxyde OH-')
plt.plot(V*1000, n_A, "-+", label = 'Ions éthanoate A-')
#plt.plot(V*1000, n_NA, "-+", label = 'Ions sodium  Na+')
plt.legend()
plt.title("Evolution des quantités de matière")
plt.xlabel("V (mL)")
```

(suite sur la page suivante)

(suite de la page précédente)

```
plt.ylabel("n (mol)")
plt.grid()

# Courbe d'évolution de la conductivité
plt.subplot(212)
plt.plot(V*1E3, sigma, "-+")
plt.axvline(Ve*1E3, color="red", linestyle ="dashed")
plt.title("Evolution de la conductivité de la solution")
plt.xlabel("V (mL)")
plt.ylabel("$\sigma~(S\cdot m^{-1})$")
plt.grid()

plt.show()
```



4.3.4 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.

4.3.4.1 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 :

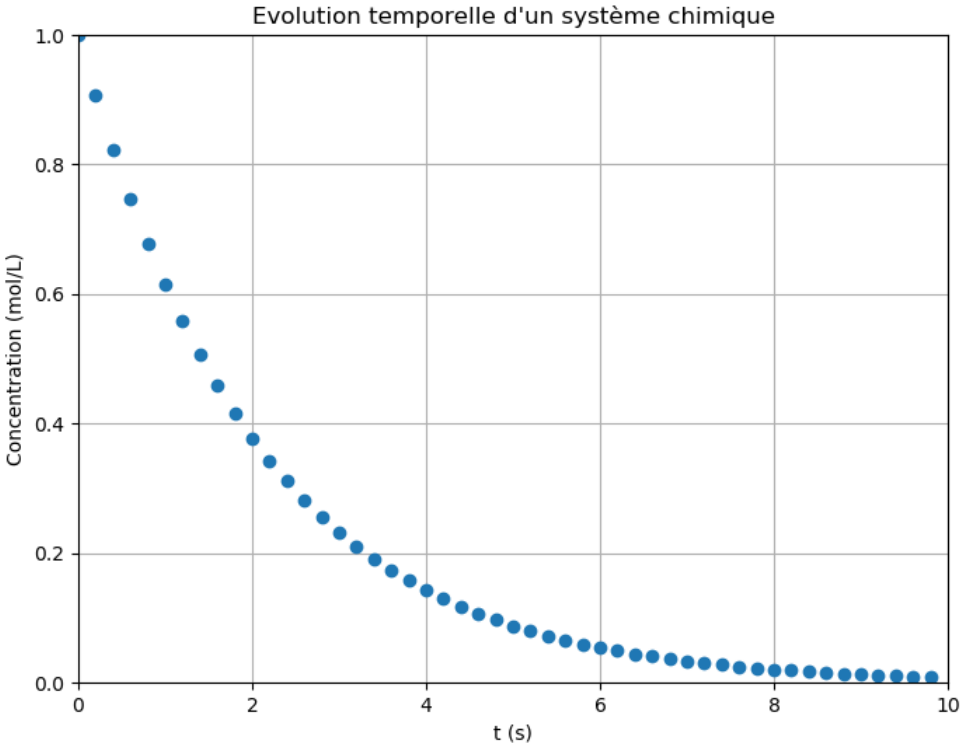
$$v_{disp}(R) = \frac{C_{n+1} - C_n}{t_{n+1} - t_n} \quad \text{en posant} \quad C = [R]$$

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()
```



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 ($v_{disp} = -k \times C_n$).

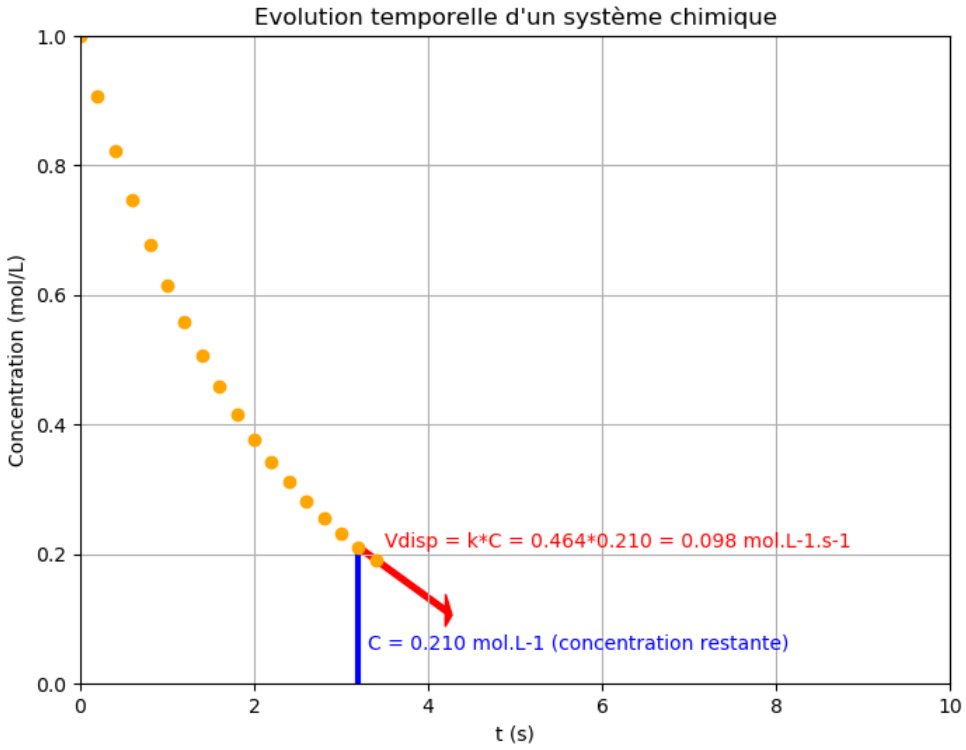
Ici une boucle *while* (avec une condition sur la concentration) est utilisée à la place d'un *for* !

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

(suite sur la page suivante)

(suite de la page précédente)

```
    # 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()
```



4.3.5 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.

4.3.5.1 Programme Python

```
A faire !
```

4.3.6 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 pK A donné.

4.3.6.1 Programme Python

```
A faire !
```

4.3.7 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.

4.3.7.1 Programme Python

```
A faire !
```

4.3.8 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.

4.3.8.1 Programme Python

```
A faire !
```

4.3.9 Les lois 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.

4.3.9.1 Programme Python

```
A faire !
```

4.3.10 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.

4.3.10.1 Programme Python

```
A faire !
```

Chapitre 5

Aller plus loin

5.1 Solution d'une équation différentielle

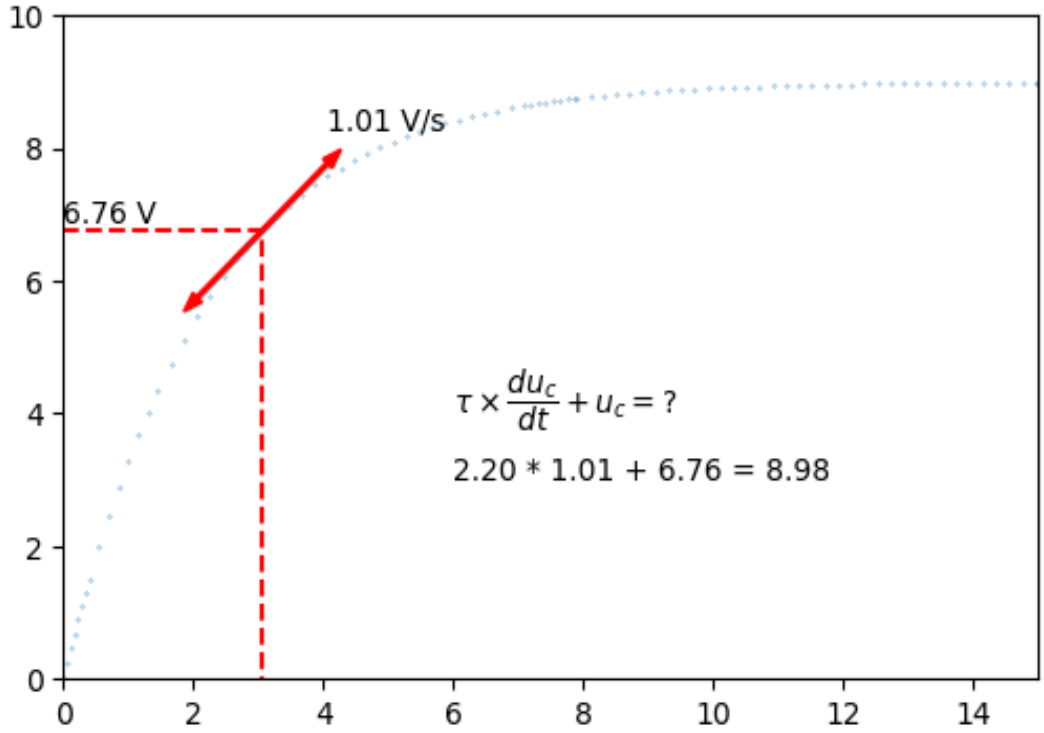
BTS Électrotechnique

Animation vérifiant la solution d'une équation différentielle du premier ordre.

Script Python

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 150

#tps,u,uc,i = np.loadtxt('data_ltspice.txt',delimiter='\t',unpack=True,skiprows=1)
tps,u,uc = np.loadtxt('data_oscillo.csv',delimiter=',',unpack=True,skiprows=2)

tau = 2.2



for n in range(10,90,1):
    a = (uc[n+1]-uc[n-1])/(tps[n+1]-tps[n-1])
    E = tau*a+uc[n]
    text1 = '%.3f V/s' % a
    text2 = '%.2f V' % uc[n]
    text3 = r'$\tau\times\dfrac{du_c}{dt} + u_c = ?$'
    text4 = '%.2f' % tau + ' * %.3f' % a + ' + %.2f' % uc[n] + ' = %.2f' % E
    plt.clf()
    plt.xlim(0,20)
    plt.ylim(0,10)
    plt.plot(tps,uc,'.',markersize=0.5)
    plt.arrow(tps[n],uc[n],1,a,color='r',width=0.05,head_width=0.2)
    plt.arrow(tps[n],uc[n],-1,-a,color='r',width=0.05,head_width=0.2)
    plt.annotate(text1,(tps[n]+1,uc[n]+a+0.5))
    plt.plot([0,tps[n],tps[n]],[uc[n],uc[n],0],'r--')
    plt.annotate(text2,(0,uc[n]+0.1))
    plt.annotate(text3,(6,4))
    plt.annotate(text4,(6,3))
    plt.pause(2)

plt.show()
```

Résultats



5.2 Micro :bit

5.2.1 Introduction

La carte Micro :bit a été développée par la BBC pour l'enseignement de l'informatique dans les écoles du Royaume Uni. Elle est programmable en MakeCode ou en MicroPython.

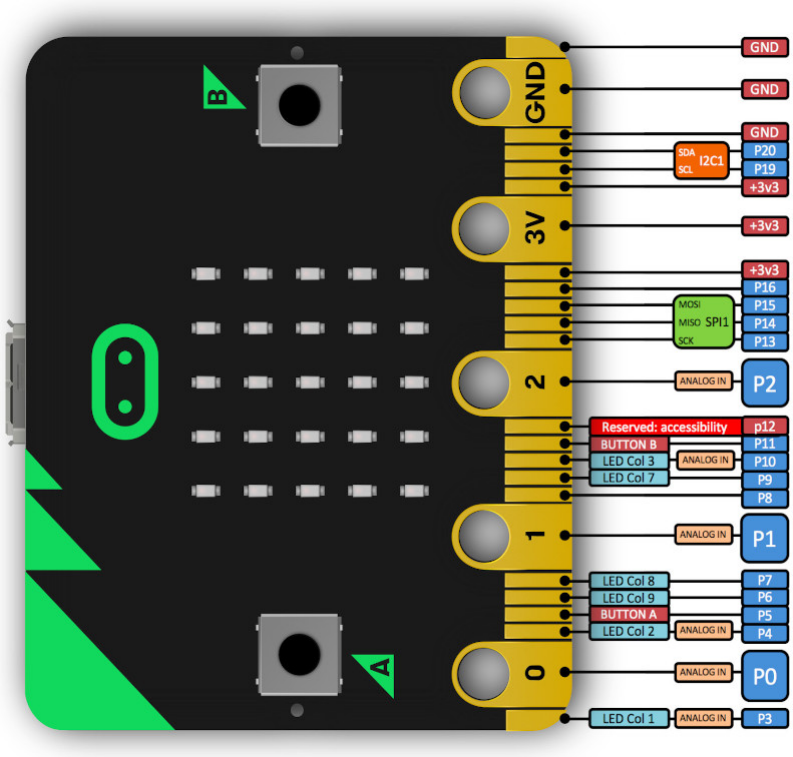
Principales caractéristiques de la carte :

- Microcontrôleur nRF51822 (ARM Cortex M0 - 32 bit - 16 Mhz) ;
- 256 ko de mémoire flash (ROM) ;
- 16 ko de mémoire vive (RAM) ;
- Tension de fonctionnement à 3,3V ;
- 1 port micro USB (programmation REPL + accés mémoire flash) ;
- 25 ports E/S ;
- 3 entrées analogique ;
- 2 boutons, matrice 5x5 leds, 1 ;

— Bluetooth 4.0 LE
— Magnétomètre 3D, accéléromètre 3D

Brochage :

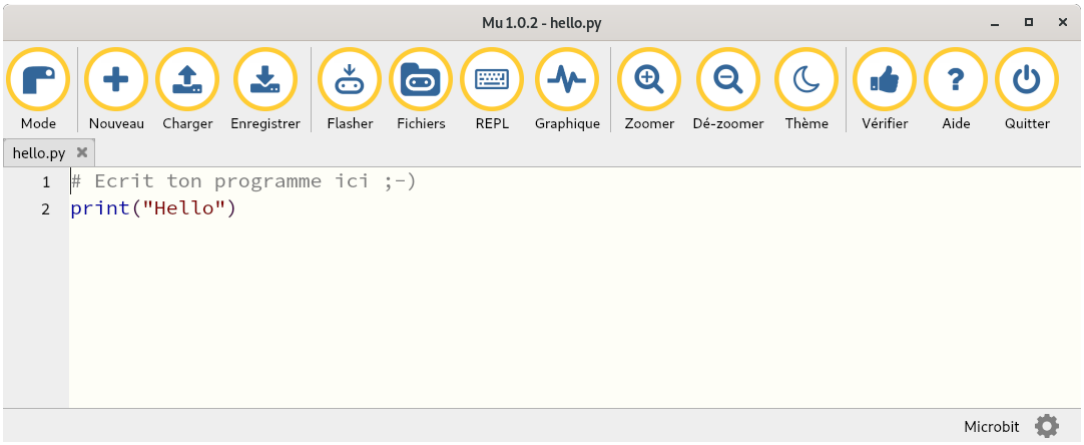
Le brochage est assez particulier. Cinq anneaux (grosses broches) donnent accès à 3 entrées/sorties (P0, P1 et P2) et à l'alimentation. Les autres ports sont accessibles sur des petites broches.



5.2.2 BBC MicroPython

BBC micro:bit MicroPython est une édition de MicroPython spécialement conçue pour Micro :bit.

Pour son développement, il est conseillé d'utiliser l'éditeur Mu.



Sinon l'éditeur Thonny est une bonne alternative à Mu.

5.2.3 Les bases

Les fonctionnalités spécifiques à la micro :bit sont gérées par librairie `microbit`.

Les broches sont notées sous la forme `pinN` où `N` est le numéro de la broche (ex. `pin0`, `pin`, ...).

5.2.3.1 Écrire sur une sortie digitale

La fonction `write_digital(val)` impose l'état logique `val` (0 ou 1) sur une sortie digitale.

```
from microbit import *
pin0.write_digital(1)   # Etat 1 sur P0
pin2.write_digital(0)   # Etat 0 sur P2
```

25 LED internes sont disposées dans une matrice 5x5.

```
???
```

5.2.3.2 Lire une entrée digitale

La fonction `read_digital()` renvoie le niveau logique sur un broche.

```
from microbit import *
val = pin0.read_digital()   # Renvoie le niveau logique sur P0
print(val)                  # Affichage du niveau logique
```

La micro :bit intègre deux boutons notés A et B respectivement avec les attributs `button_a` et `button_b` . Ils sont connectés à P5 et P11.

```
from microbit import *
val = button_a.is_pressed() # renvoie True ou False
print(val)                  # Affichage de l'état du bouton A
```

5.2.3.3 Générer une tension MLI (PWM)

Comme avec Arduino, il est possible de générer une tension Modulée en Largeur d'Impulsion (MLI ou PWM en anglais) avec la fonction `write_analog(duty)`. Le paramètre `duty` est le rapport cyclique codé sur 12 bits (de 0 à 1023 pour un rapport cyclique de 0 à 100%).

La fréquence du signal est fixée par les fonctions `set_analog_period(T)` ou `set_analog_period_microseconds(T)` où `T` est la période respectivement en millisecondes et microsecondes.

```
from microbit import *
Pin0.set_analog_period(100)  # fixe une période de 100 ms
pin0.write_analog(767)       # rapport cyclique à 75% sur P0
```

Note : Il est intéressant ici de mesurer la tension moyenne au voltmètre numérique (entre GND et P0) en position DC.

5.2.3.4 Mesurer une tension (CAN)

La lecture sur 12 bits d'une tension entre 0 V et 3,3 V est effectuée par la méthode `read_analog()`.

```
from microbit import *
val = Pin0.read_analog() # renvoie un nombre entre 0 à 1023
print(val*3.3/1023)      # affichage de la tension
```

5.2.3.5 Générer une tension analogique (CNA)

La carte ne dispose pas de vraies sorties analogiques (pas de CNA) !

5.2.3.6 Faire une pause

Les fonctions `sleep(T)`, `sleep_ms(T)` et `sleep_us(T)` du module `utime` permettent de faire une pause de durée `T` respectivement en seconde, milliseconde et microseconde.

```
from utime import sleep
while True:
   Pin0.write_digital(1)
   sleep(1)
   Pin0.write_digital(0)
   sleep(1)
```

5.2.3.7 Mesurer une durée

Il est possible de mesurer la durée d'une impulsion à l'état haut ou l'état bas avec la fonction `time_pulse_us()` du module `machine` commun à tous les microcontrôleurs sous MicroPython.

exemple mesurer la durée à l'état haut d'une impulsion sur l'entrée X1.

```
from microbit import *
from machine import time_pulse_us
duree = time_pulse_us(Pin0,1)
print(duree)
```

Chapitre 6

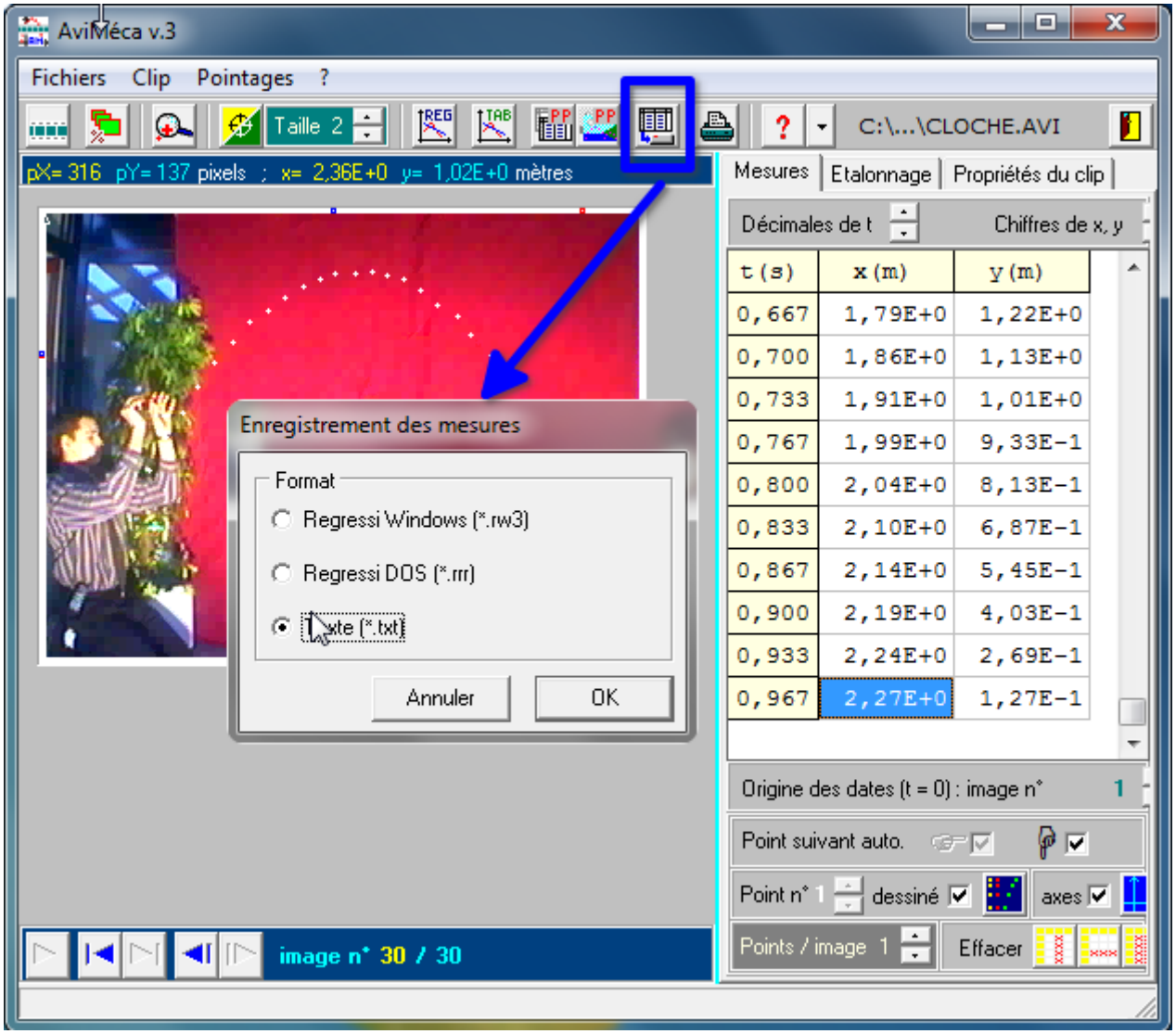
Annexes

6.1 Importer dans Python des données à partir d'AviMeca

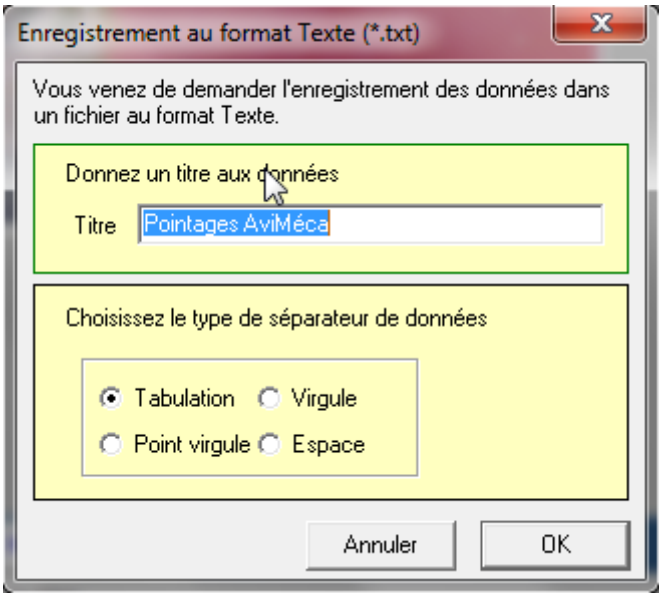
AviMeca, comme le plupart des logiciels utilisés en physique-chimie, offre la possibilité d'exporter les mesures dans un fichier texte (avec l'extension `.txt` au format CSV). Il est ainsi possible d'exploiter directement les données dans Python sans avoir à les recopier manuellement dans le script !

6.1.1 Étape 1 : Exporter les mesures dans un fichier texte

Une fois les mesures effectuées, cliquer sur l'icône d'enregistrement de mesures (voir figure ci-dessous) et choisir `Texte (*.txt)`.



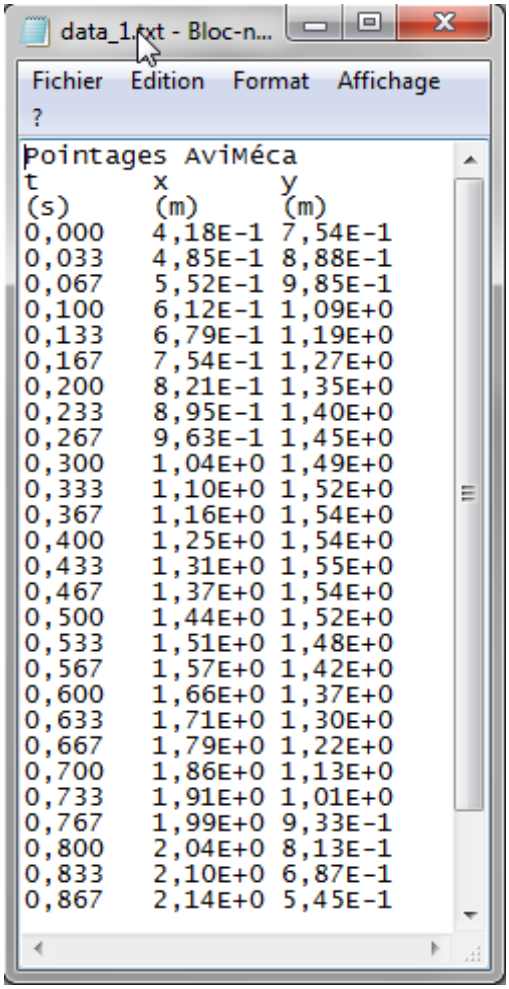
Dans la fenêtre suivante, ajouter un titre (éviter les accents !) et garder la tabulation comme séparateur (plus lisible).



Puis enregistrer le fichier (par exemple avec le nom `data.txt`).

6.1.2 Étape 2 : Adapter le fichier

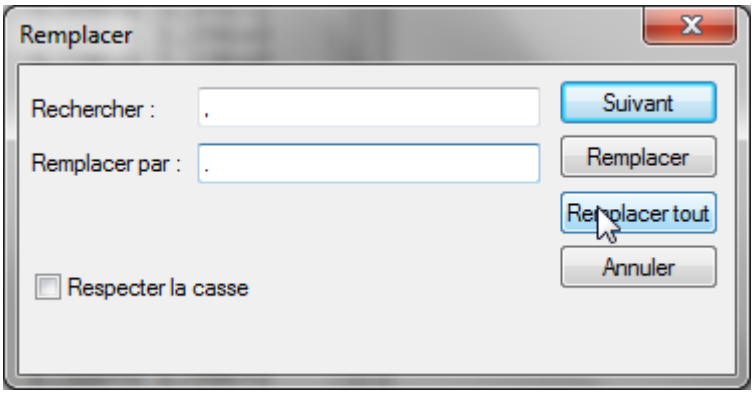
Ouvrir le fichier obtenu avec un éditeur texte (exemple. Bloc-note, Wordpad, Wordpad++...).



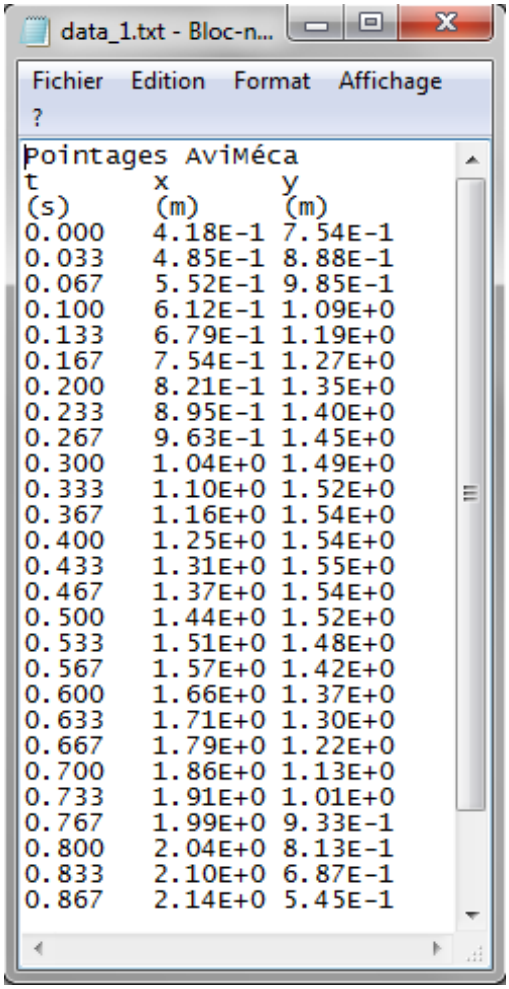
Il faut maintenant adapter les nombres décimaux au système anglo-saxon, c'est à dire remplacer toutes les virgules par des points !

Dans la boite de dialogue « Remplacer » (dans le menu édition) :

- compléter le champ « Rechercher » avec le caractère **,** ;
- le champ « Remplacer par » avec le caractère **.** ;
- puis cliquer sur « Remplacer tout ».



On obtient alors le contenu suivant :



Enregistrer le fichier pour prendre en compte les modifications.

6.1.3 Étape 3 : Importer les données dans Python

La fonction `loadtxt` du module `numpy` se chargera d'importer les données du fichier `data.txt` dans Python.

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

D'après l'analyse du fichier texte, il faudra ajouter les options suivantes à la fonction `loadtxt()` :

`delimiter ='\t'`	Tabulation comme séparateur.
`skiprows=1`	Sauter les trois premières lignes (commentaires).
`unpack=True`	Les données sont en colonnes (en lignes par défaut).

Ce qui donne pour la fonction `loadtxt()` :

```
t, x, y = np.loadtxt('data.txt',delimiter='\t',skiprows=3, unpack=True)
```

Et voici le programme complet :

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

t, x, y = np.loadtxt('data.txt',delimiter='\t',skiprows=3, unpack=True)
```

(suite sur la page suivante)

(suite de la page précédente)

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

Et la courbe obtenue :