LA PROGRAMMATION PYTHON

- Nous traiterons dans cette fiche technique plusieurs exemples dont deux principaux : étude de la caractéristique d'une résistance et étude du mouvement d'un système.
- Les noms des fichiers contenant les programmes Python seront systématiquement précisés dans le titre des documents et en-tête du programme.
- Chaque programme sera suivi du résultat de son exécution.

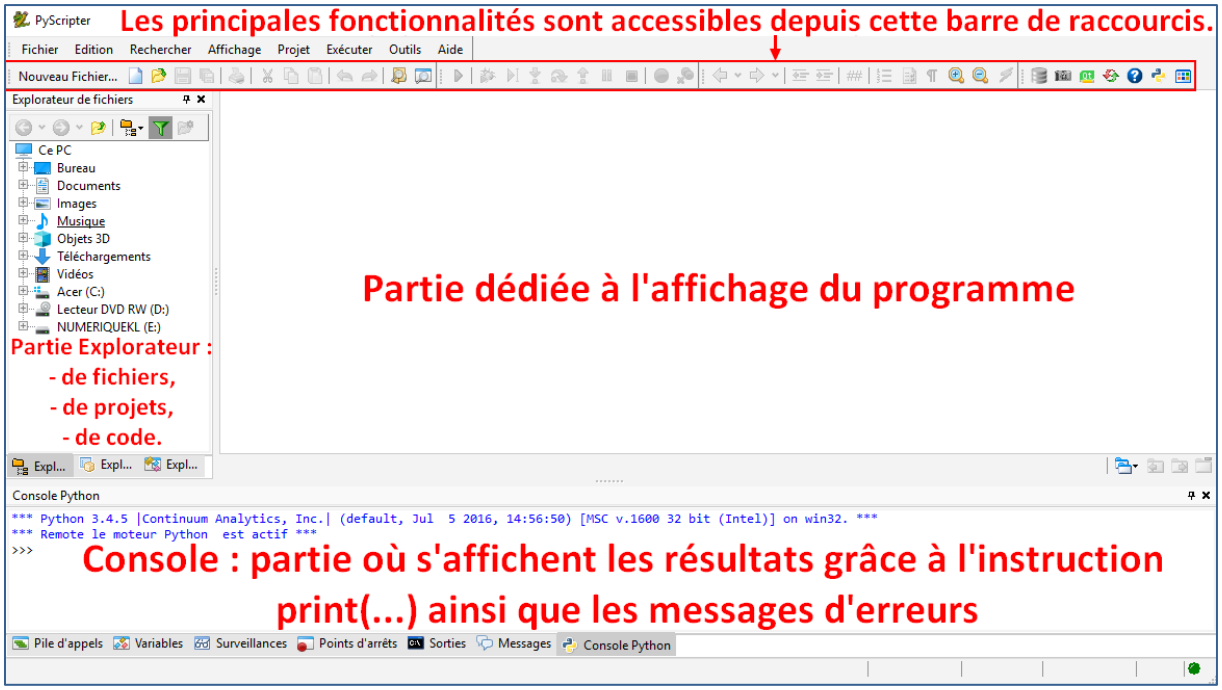
A. L'environnement de développement

1. Le logiciel de programmation utilisé

- ***EduPython :*** Principalement conçu pour la programmation avec les élèves, cet environnement de développement est à la fois simple et complet. Il est téléchargeable à l'adresse suivante : https://edupython.tuxfamily.org/

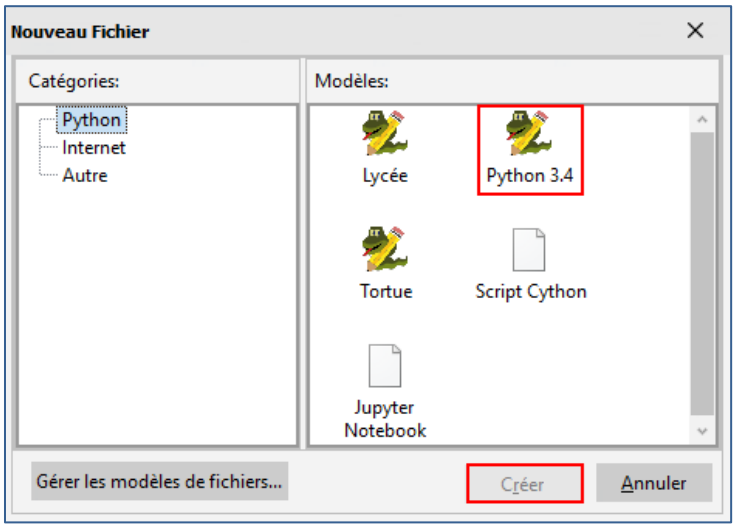
Doc. 1 EduPython

Doc. 2 Aperçu du logiciel lors de son ouverture



• Pour créer un nouveau programme, il est nécessaire de cliquer sur « Nouveau fichier » dans la barre de raccourcis puis sur Python3.x (en fonction de votre version) (Doc 3). Le raccourci CTRL+N ou l'icône  pourront également être utilisés pour créer rapidement un document sans nécessairement l'enregistrer.
• Le nom du fichier d'extension .py ne doit pas comporter d'espace, d'accent, de caractère spécial. Il commencera par une minuscule (Ex. etudeLoiOhm.py). Le nom du fichier ne devra pas porter le nom d'une bibliothèque utilisée (voir paragraphe A.2).

Doc. 3 Création d'un nouveau fichier



• Il est fort probable que l'environnement de travail soit noir avec une saisie du texte dans des tons clairs. C'est en effet ce qui est préconisé pour les personnes qui travaillent toute la journée devant un écran. Néanmoins, il est plus facile pour faire des copies d'écran, et éventuellement des impressions, de travailler avec un fond clair. Il est possible de changer le style avec l'icône ci-dessous.

Doc. 4 Modification du style

• Il est fréquent de fermer la fenêtre d'exploration ou la console Python, pour réinitialiser l'interface et les options vous pouvez utiliser l'icône suivante.

Doc. 5 Réinitialisation de l'interface et des options

• Les 2 dernières icônes indispensables pour bien démarrer sont :
- le triangle vert pour exécuter un programme (Doc 6),
- le carré rouge pour l'arrêter en cas de problème (Doc 7).

Doc. 6 Exécution d'un programme

Doc. 7 Arrêt du processus en cas de problème

2. Les bibliothèques nécessaires

• Une des grandes forces du langage Python réside dans le nombre important de bibliothèques externes disponibles. Une bibliothèque, également appelée librairie, est un ensemble de fonctions qui permettent de faire :

- des calculs => "math",
- des graphiques => "matplotlib",
- de la programmation de jeux => "pygame",
- de la gestion des images => "PIL",
- de la gestion d'un microcontrôleur => "serial",
- ...

• Les bibliothèques courantes sont disponibles dans EduPython. Selon les objectifs visés, il est nécessaire d'importer tout ou une partie des bibliothèques correspondantes en tout début de programme.

Doc. 8 Exemples d'import

```
import matplotlib.pyplot as plt
import serial as sr
import math as mt
from math import pi, cos, sin
```

Par exemple, dans le 1er cas, nous importons le module "pyplot" de la bibliothèque "matplotlib" sous l'alias "plt". Dans les 2 cas suivants, nous importons toute la bibliothèque en utilisant un alias. Dans le dernier cas, nous importons quelques fonctions de la bibliothèque "math" (Doc 8).

B. Quelques notions de programmation

1. L'indentation et les commentaires

• L'indentation, c'est-à-dire l'ajout de quatre espaces ou d'une tabulation en début de ligne, est primordiale en Python, elle fait partie intégrante du langage.
• Elle permet d'indiquer les lignes de code incluses dans un bloc d'instructions tel qu'une boucle, une méthode. Il n'y a pas d'accolade comme en Arduino ou dans beaucoup d'autres langages (voir paragraphe B.4 pour des exemples).
• Il faut par conséquent être vigilant à ne pas introduire d'espace en début de ligne.

• Les commentaires sont indispensables pour expliquer le code ou garder les essais et les premiers tests.
• Ils peuvent s'écrire sur une ligne grâce à un #, sur plusieurs lignes en commençant et terminant par """ (triple-guillemets) ou ''' (triple-cote) (Doc 9).

2. Les variables

• Elles sont nécessaires pour stocker et exploiter les mesures, les paramètres, les valeurs initiales.
• Il suffit d'écrire le nom de la variable, le symbole = et la valeur que nous souhaitons affecter (Doc 9). Il est d'usage de laisser un espace entre le nom de la variable, le symbole = et la valeur.
• Le nom des variables doit être explicite, il commence par une lettre de préférence en minuscule. Pour des noms composés, le guide de style préconise deux méthodes : le second mot est écrit avec une majuscule (Ex. tensionMoteur) ou les mots du nom sont séparés par un tiret bas (tiret du 8) ou « underscore » (Ex. tension_moteur, temps_ms).
• Le logiciel EduPython est doté de l'auto-complétion : une fois la variable créée, son nom sera proposé en tapant les premières lettres, il ne faut donc pas craindre d'utiliser de longs noms.

Doc. 9 Exemple de variables (variables.py)

```
'''
Fichier variables.py
'''
"""
Exemple sur l'affectation des variables, l'affichage
et la saisie par l'utilisateur
"""
# AFFECTATION DE VARIABLES
resistance = 220        # Un entier => int
intensite = 0.031       # Un nombre décimal écrit avec un point => float
tension = resistance * intensite       # tension sera de type float
# AFFICHAGE DE MESSAGE
print ("Premier test de Python")
# AFFICHAGE DE VARIABLES
print(resistance)
print(intensite)
print(tension)
# AFFICHAGE SIMULTANE DE VARIABLES ET DE MESSAGE
print ("La tension est le produit de",resistance,"par",intensite,"soit :",tension)
# DEMANDE A L'ULTISATEUR DE SAISIR UNE VARIABLE
nom = input("Indiquer le nom du TP :")          # Du type chaîne de caractères
poste = int(input("Préciser votre poste :"))    # Du type entier
note = float(input("Indiquer votre note :"))    # Du type décimal
```

```
Premier test de Python
220
0.031
6.82
La tension est le produit de 220 par 0.031 soit : 6.82
```

3. Les listes de données

• Les listes sont indispensables pour l'étude des séries de mesures, les représentations graphiques.
• Pour distinguer les variables des listes de valeurs, il est recommandé de précéder le nom de la grandeur de 'list' (ou 'tab' pour tableau). Mais en mécanique, cela

présente l'inconvénient de rallonger les expressions des calculs de vitesse et des tracés de vecteurs. Dans ce cas, il sera possible de noter x et y plutôt que list_x et list_y.

• Chaque valeur d'une liste est repérée par un indice commençant à 0. Pour atteindre la nième valeur, il faut prendre l'habitude d'utiliser l'indice n-1 (Doc 10).

Doc. 10 Exemple de création et manipulation de listes (etudeResistance.py)

```
'''
Fichier etudeResistance.py
'''
# Création d'un listleau à partir d'une liste de valeurs
list_U_V = [0,0.5,1,1.5,2,2.5,3,3.5,4,4.5]
list_I_A = [0,10.6E-3,21.3E-3,32.9E-3,42.5E-3,53.1E-3,63.8E-3,74.5E-3,85.1E-3,95.7E-3]
# Ajout d'une valeur à la fin du listleau
list_U_V.append(5)
list_I_A.append(0.106)
# Affichage de la troisième valeur du listleau list_U_V
print("list_U_V[2] =",list_U_V[2])
# Affichage des 4 premières valeurs de list_I_A (indice 0 à 3)
print("list_I_A[0:4] =",list_I_A[0:4])
# Affichage de la taille d'un listleau
print("len(list_U_V) =",len(list_U_V))
# Affichage de toutes les valeurs d'un listleau
for i in range(0, len(list_U_V)) :
    print("Pour U =",list_U_V[i],"V, on obtient I =",list_I_A[i],"A.")
# Modification de la 3ème valeur de list_I_A
list_I_A[2] = 21.2E-3
```

```
list_U_V[2] = 1
list_I_A[0:4] = [0, 0.0106, 0.0213, 0.0329]
len(list_U_V) = 11
Pour U = 0 V, on obtient I = 0 A.
Pour U = 0.5 V, on obtient I = 0.0106 A.
Pour U = 1 V, on obtient I = 0.0213 A.
Pour U = 1.5 V, on obtient I = 0.0329 A.
Pour U = 2 V, on obtient I = 0.0425 A.
Pour U = 2.5 V, on obtient I = 0.0531 A.
Pour U = 3 V, on obtient I = 0.0638 A.
Pour U = 3.5 V, on obtient I = 0.0745 A.
Pour U = 4 V, on obtient I = 0.0851 A.
Pour U = 4.5 V, on obtient I = 0.0957 A.
Pour U = 5 V, on obtient I = 0.106 A.
```

• Nous verrons dans le paragraphe C.6 comment récupérer les valeurs obtenues lors d'un pointage vidéo sous forme de listes dans EduPython.

4. Les tests

• Nous utiliserons les tests 'si … alors (… sinon)' avec les microcontrôleurs mais également en Python. Prenons comme exemple la détermination de l'avancement maximal de la réaction suivante :

$$I_{2(aq)} + 2\ S_2O_3^{2-}{}_{(aq)} \rightarrow 2\ I^-_{(aq)} + S_4O_6^{2-}{}_{(aq)}$$

Doc. 11 Détermination de l'avancement maximal (tests.py)

```
'''
Fichier tests.py
'''
# INSTRUCTION CONDITIONNELLE
# Réaction entre le diiode et les ions thiosulfate
nDiiode = 0.012 # en mol
nIonThiosulfate = 0.017 # en mol
xmax1 = nDiiode
xmax2 = nIonThiosulfate / 2
if (xmax1 < xmax2) :
    print("Le réactif limitant est le diiode.")
    xmax = xmax1
else :
    print("Le réactif limitant est l'ion thiosulfate.")
    xmax = xmax2
print("L'avancement maximal est donc xmax =", xmax, "mol.")
```

```
Le réactif limitant est l'ion thiosulfate.
L'avancement maximal est donc xmax = 0.0085 mol.
```

5. Les boucles

• Il existe plusieurs types de boucles. La plus couramment utilisée dans les activités de physique-chimie est la boucle bornée afin de gérer, par exemple, les calculs de vitesses et les tracés de vecteurs vitesse.

• Nous l'appelons également la boucle 'pour' (en algorithmique) ou boucle 'for'. Dans l'exemple ci-dessous, nous lisons : pour i allant de 1 à 10 (la dernière valeur n'est pas atteinte), afficher la phrase ...

Doc. 12 Exemple de boucle 'Pour' ou 'For' (boucles.py)

```
'''
Fichier boucles.py
'''
# BOUCLE ITERATIVE POUR
for i in range (1,11) :
    print("n°",i,"Je dois être attentif(ve) en classe.")
```

```
n° 1 Je dois être attentif(ve) en classe.
n° 2 Je dois être attentif(ve) en classe.
n° 3 Je dois être attentif(ve) en classe.
n° 4 Je dois être attentif(ve) en classe.
n° 5 Je dois être attentif(ve) en classe.
n° 6 Je dois être attentif(ve) en classe.
n° 7 Je dois être attentif(ve) en classe.
n° 8 Je dois être attentif(ve) en classe.
n° 9 Je dois être attentif(ve) en classe.
n° 10 Je dois être attentif(ve) en classe.
```

• En calculant la vitesse au point M_i, le problème rencontré lors de la programmation est de gérer des listes de tailles différentes : pour N points, nous obtenons N-1 valeurs de vitesse. Dans la boucle 'for' il faudra veiller à bien modifier la borne supérieure lors du calcul des vitesses et de l'affichage des vecteurs.

Doc. 13 Exemple pour calculer les coordonnées des vecteurs vitesse (mouvementBalle.py)

```
'''
Fichier mouvementBalle.py
'''
x = [0.96, 1.04, 1.12, 1.21, 1.28, 1.37, 1.46, 1.53, 1.61, 1.68, 1.77] # en m
y = [1.16, 1.26, 1.35, 1.43, 1.49, 1.56, 1.62, 1.68, 1.72, 1.76, 1.81] # en m
dt = 0.04 # en s
vx = [] # en m/s
vy = [] # en m/s
nbrePoints = len(x)

for i in range(0,nbrePoints-1) :
    vx.append((x[i+1]-x[i])/dt)
    vy.append((y[i+1]-y[i])/dt)
```

• Pour finir, il existe également la boucle non bornée 'tant que' ou 'while', c'est une boucle itérative qui s'exécute tant que la condition précisée est vraie. Nous l'utilisons lorsque nous ne savons pas par avance le nombre de répétitions. Elle nécessite une vigilance particulière car si la condition exprimée reste vraie en permanence le programme boucle à l'infini, ce qui oblige à fermer EduPython.

Doc. 14 Exemple de boucle 'Tant que' ou 'While' (boucles.py)

```
'''
Fichier boucles.py
'''
# BOUCLE NON BORNEE TANT QUE
import random as rd
# Bibliothèque pour gérer les nombres aléatoires
# Ci-dessous un nombre entier entre 1 et 100 est choisi aléatoirement
nombreMystere  = rd.randint(1,100)
nombrePropose = int(input("Entrer un nombre compris entre 1 et 100 :"))
while (nombrePropose != nombreMystere):
    if (nombrePropose > nombreMystere) :
        print("Trop grand !")
    if (nombrePropose < nombreMystere) :
        print("Trop petit !")
    nombrePropose = int(input("Entrer un nombre compris entre 1 et 100 :"))
print("Vous avez gagné !")
```

C. La bibliothèque "matplotlib"

• "matplotlib" est l'une des bibliothèques Python les plus utilisées pour représenter des graphiques en 2D. Elle permet de produire une grande variété de graphiques de grande qualité.

• Le module "pyplot" de "matplotlib" est l'un de ses principaux modules. Il regroupe un grand nombre de fonctions qui servent à créer des graphiques et les personnaliser (possibilité de travailler sur les axes, le type de graphique, sa forme et même rajouter du texte).

1. Les nuages de points

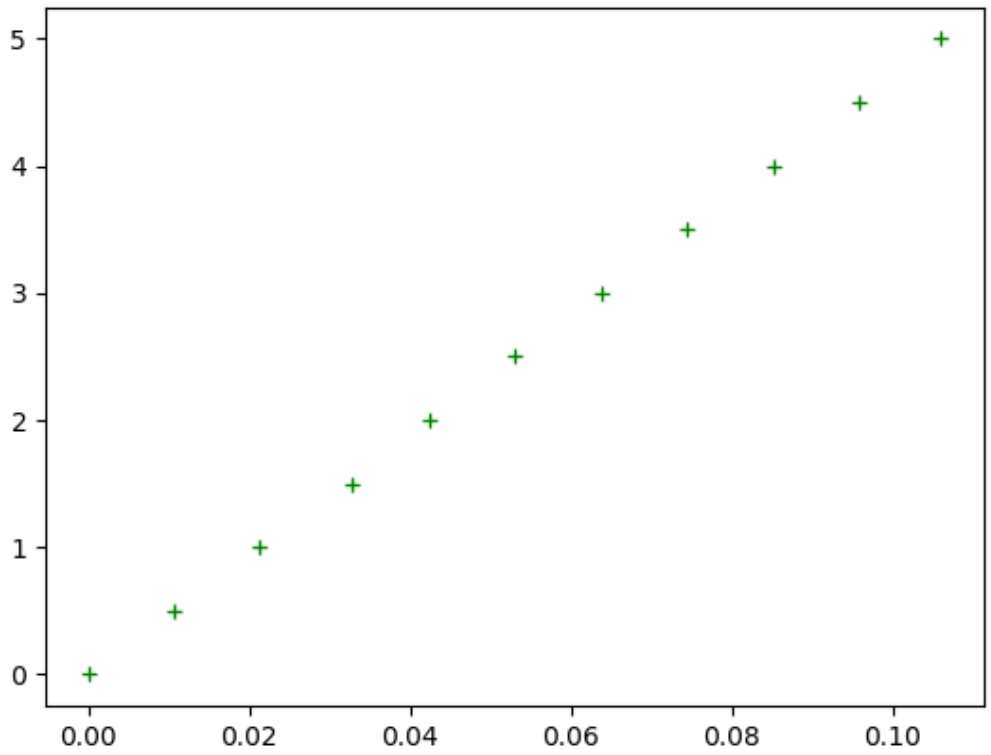
• Nous allons reprendre l'exemple de la résistance et des mesures de tensions et d'intensités du courant électrique.

Doc. 15 Représentation graphique U en fonction de I (etudeResistance.py)

```
'''
Fichier etudeResistance.py
'''
import matplotlib.pyplot as plt

list_U_V = [0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5.0]
list_I_A = [0,10.6E-3,21.3E-3,32.9E-3,42.5E-3,53.1E-3,63.8E-3,74.5E-3,85.1E-3,95.7E-3,106E-3]

# Pour tracer un nuage de points avec les valeurs contenues dans les 2 listes.
plt.plot(list_I_A,list_U_V, 'g+')
# Pour afficher le graphique, toujours en dernier.
plt.show()
```



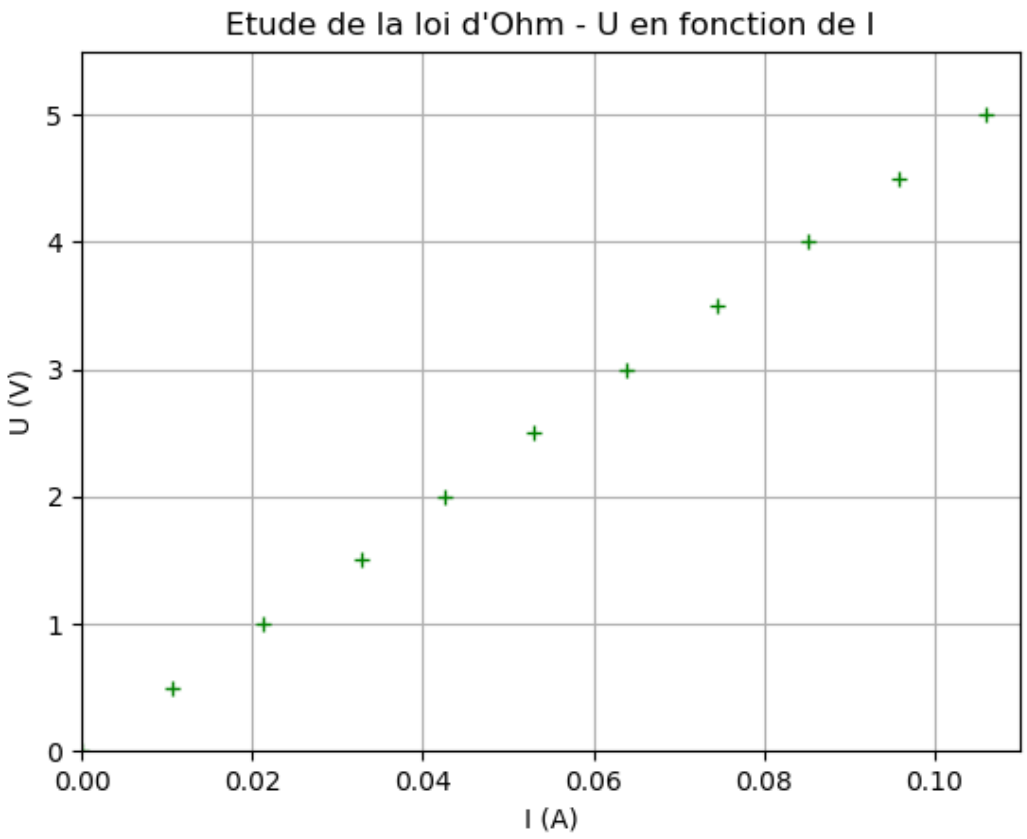
• Nous obtenons bien un nuage de points représentant l'évolution de U en fonction de I, les croix vertes ont été obtenues grâce à l'instruction 'g+', g pour green et + pour la forme des points. Il manque encore des éléments importants : grandeurs et unités, titre, éventuellement un quadrillage, des axes qui commencent à 0, …

Doc. 16 Amélioration de l'exemple précédent (etudeResistance.py)

```
'''
Fichier etudeResistance.py
'''
import matplotlib.pyplot as plt

list_U_V = [0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5.0]
list_I_A = [0,10.6E-3,21.3E-3,32.9E-3,42.5E-3,53.1E-3,63.8E-3,74.5E-3,85.1E-3,95.7E-3,106E-3]

# Pour tracer un nuage de points avec les valeurs contenues dans les 2 listes.
plt.plot(list_I_A,list_U_V, 'g+')
# Pour indiquer les grandeurs et unités en abscisse et ordonnée.
plt.xlabel("I (A)")
plt.ylabel("U (V)")
# Pour préciser un titre au graphique.
plt.title("Etude de la loi d'Ohm - U en fonction de I")
# Pour afficher un quadrillage.
plt.grid(True)
# Pour fixer les valeurs minimale et maximale sur chaque axe.
plt.axis([0, 0.11, 0, 5.5])
# Pour afficher le graphique, toujours en dernier.
plt.show()
```



2. Les courbes

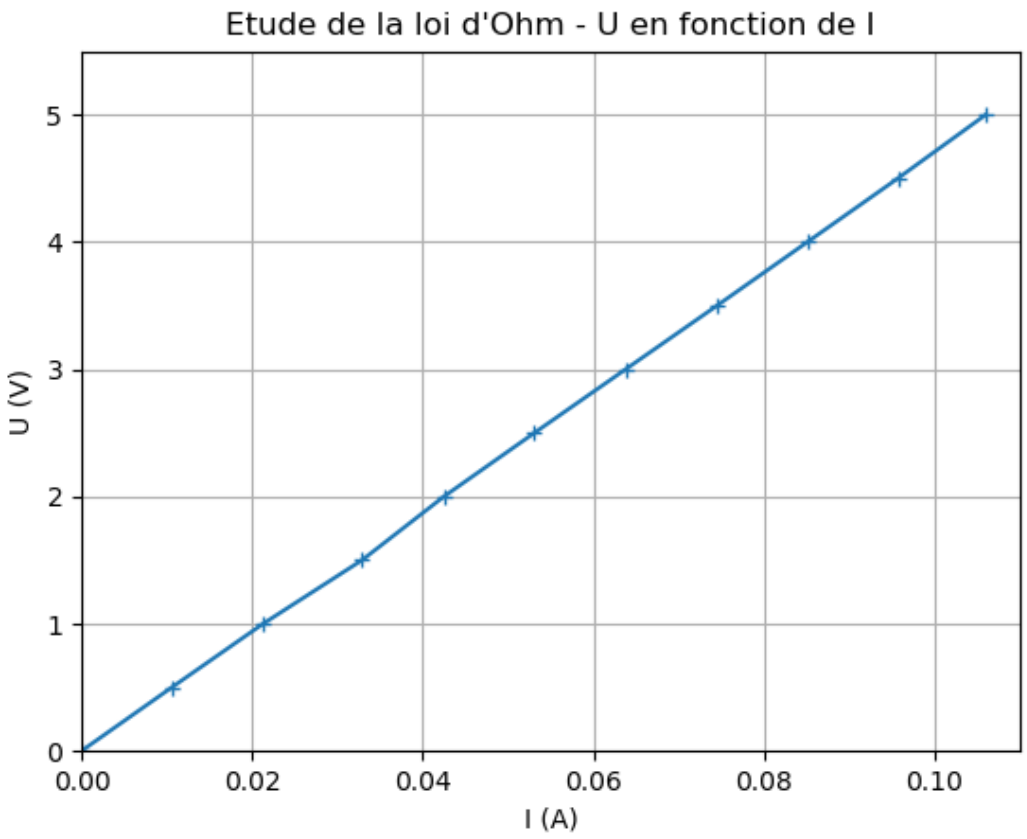
- La méthode *plot()* permet également de relier les points par un segment de droite.

Doc. 17 Modification de l'exemple précédent en modifiant les paramètres de *plot()*

```
'''
Fichier etudeResistance.py
'''
import matplotlib.pyplot as plt

list_U_V = [0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5.0]
list_I_A = [0,10.6E-3,21.3E-3,32.9E-3,42.5E-3,53.1E-3,63.8E-3,74.5E-3,85.1E-3,95.7E-3,106E-3]

# Pour tracer les segments de droite entre les points
plt.plot(list_I_A,list_U_V, marker = '+')
# Pour indiquer les grandeurs et unités en abscisse et ordonnée.
plt.xlabel("I (A)")
plt.ylabel("U (V)")
# Pour préciser un titre au graphique.
plt.title("Etude de la loi d'Ohm - U en fonction de I")
# Pour afficher un quadrillage.
plt.grid(True)
# Pour fixer les valeurs minimale et maximale sur chaque axe.
plt.axis([0, 0.11, 0, 5.5])
# Pour afficher le graphique, toujours en dernier.
plt.show()
```



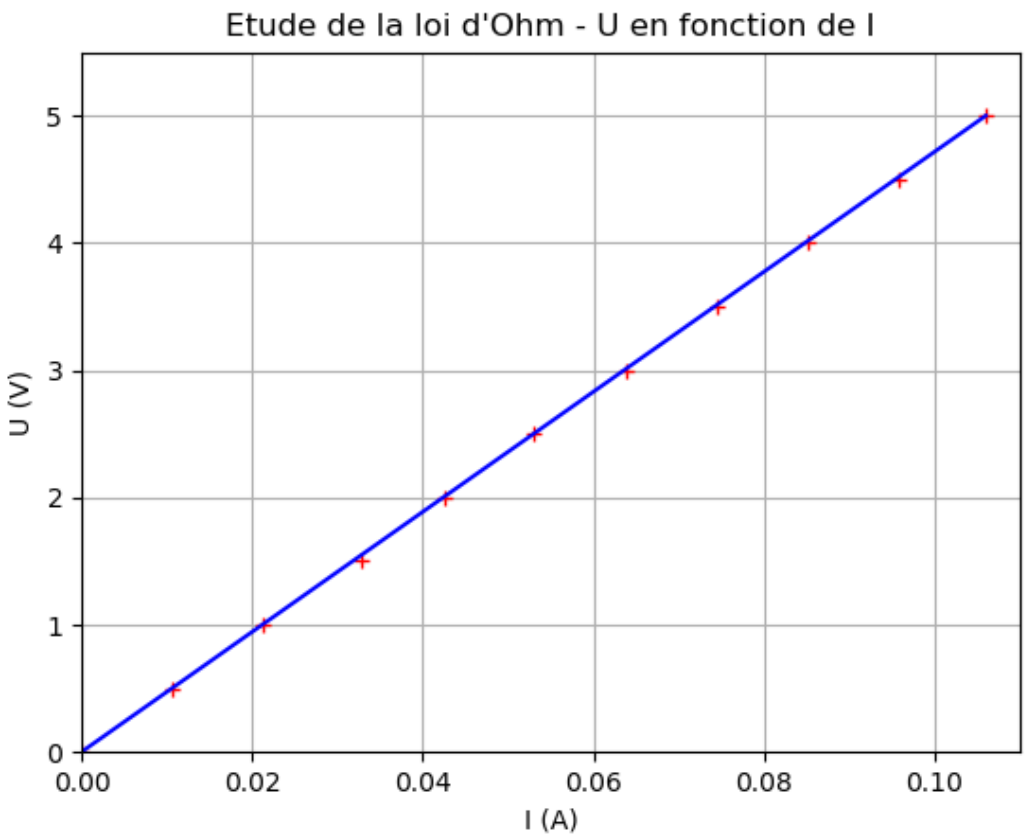
- Nous pouvons par conséquent utiliser la méthode *plot()* pour tracer la droite passant au milieu du nuage de points comme nous le ferions à la règle. Il faut estimer les points extrêmes qui semblent définir la droite.
- Nous verrons dans la partie D comment obtenir la régression linéaire correspondante.

Doc. 18 Tracé de la droite passant au milieu du nuage de points (etudeResistance.py)

```
'''
Fichier etudeResistance.py
'''
import matplotlib.pyplot as plt

list_U_V = [0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5.0]
list_I_A = [0,10.6E-3,21.3E-3,32.9E-3,42.5E-3,53.1E-3,63.8E-3,74.5E-3,85.1E-3,95.7E-3,106E-3]

# Pour tracer les segments de droite entre les points
plt.plot(list_I_A,list_U_V, 'r+')
# Pour indiquer les grandeurs et unités en abscisse et ordonnée.
plt.xlabel("I (A)")
plt.ylabel("U (V)")
# Pour préciser un titre au graphique.
plt.title("Etude de la loi d'Ohm - U en fonction de I")
# Pour afficher un quadrillage.
plt.grid(True)
# Pour fixer les valeurs minimale et maximale sur chaque axe.
plt.axis([0, 0.11, 0, 5.5])
# Pour tracer la droite passant au milieu du nuage de points.
plt.plot([0, 0.106], [0, 5], c = 'blue')
# Pour afficher le graphique, toujours en dernier.
plt.show()
```



3. Les vecteurs

• La représentation des vecteurs est possible grâce à la méthode *arrow()*. Il est nécessaire d'indiquer à l'intérieur de la parenthèse dans l'ordre suivant :

- L'abscisse du point de départ du vecteur,

- L'ordonnée du point de départ du vecteur,
- Sa longueur sur l'axe des abscisses,
- Sa longueur sur l'axe des ordonnées,
- Sa couleur,
- La largeur de la tête de la flèche,
- Le fait que la mesure inclue ou non la tête de la flèche.

• Il est indispensable de choisir une échelle lorsque la norme du vecteur représente une autre grandeur que celles des axes, par exemple, pour faire apparaître les vecteurs vitesse sur une trajectoire.

Doc. 19 Représentation des vecteurs vitesse à partir du doc 12 (mouvementBalle.py)

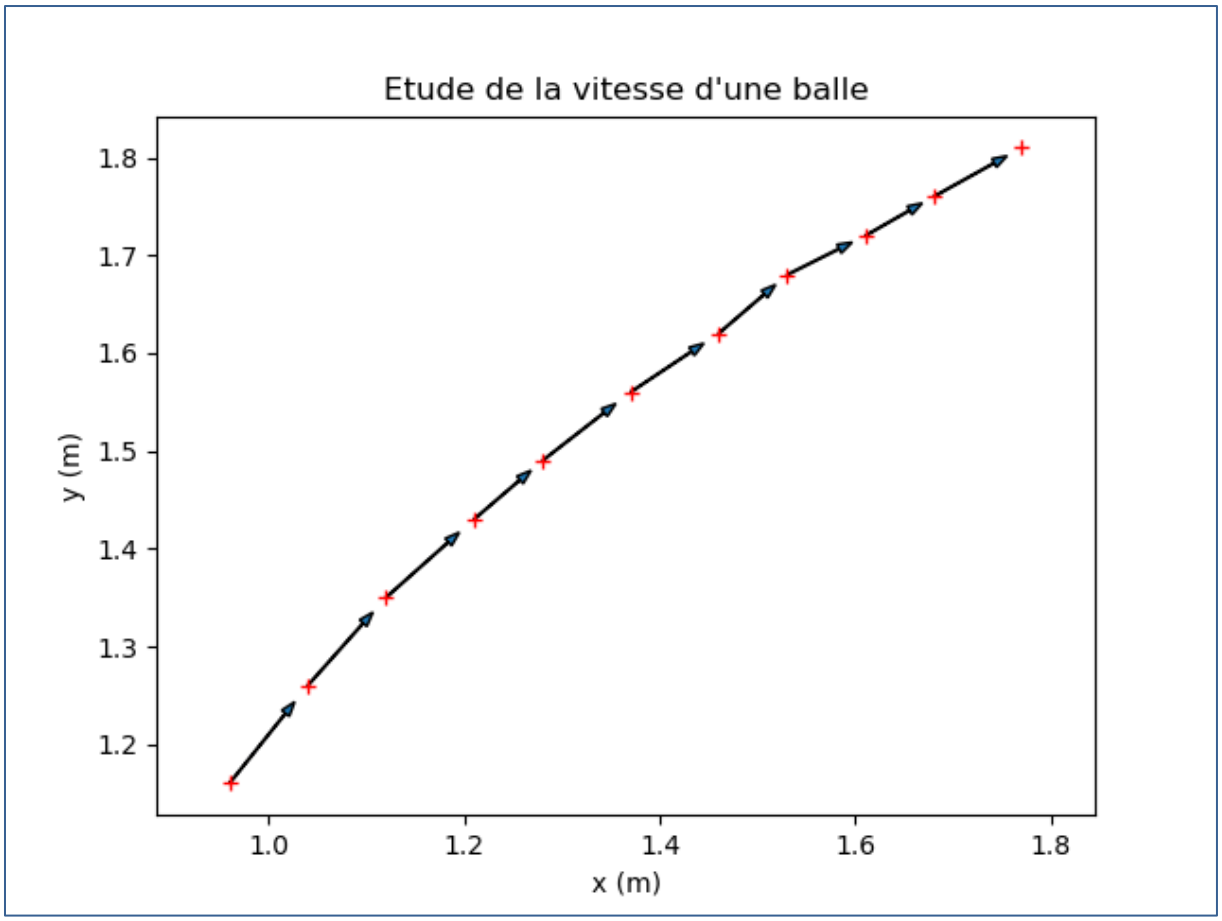
```
'''
Fichier mouvementBalle.py
'''
import matplotlib.pyplot as plt

"""
Programme du doc 13
"""

# Pour afficher les valeurs maximales et choisir judicieusement l'échelle
print("La valeur maximale pour vx est",max(vx),"m/s.")
print("La valeur maximale pour vy est",max(vy),"m/s.")
echelle = float(input("Choisir une échelle : 1 m pour ... m/s.")) # Valeur choisie : 30

for i in range(0,nbrePoints-1) :
    # Pour tracer les vecteurs vitesse aux points Mi
    plt.arrow(x[i], y[i], vx[i]/echelle, vy[i]/echelle, head_width = 0.01, length_includes_head = True)

plt.plot(x, y, 'r+')
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.title("Etude de la vitesse d'une balle")
plt.axis('equal') # Pour avoir un repère orthonormé
plt.show()
```



4. Les annotations

• Il est possible d'ajouter des annotations ou du texte sur les représentations graphiques grâce aux instructions *text()* et *annote()*.

• Pour utiliser la méthode *text(),* il faut préciser l'abscisse et l'ordonnée du point de départ du texte puis le texte. Il est possible également de modifier, entre autres, la taille de la police.

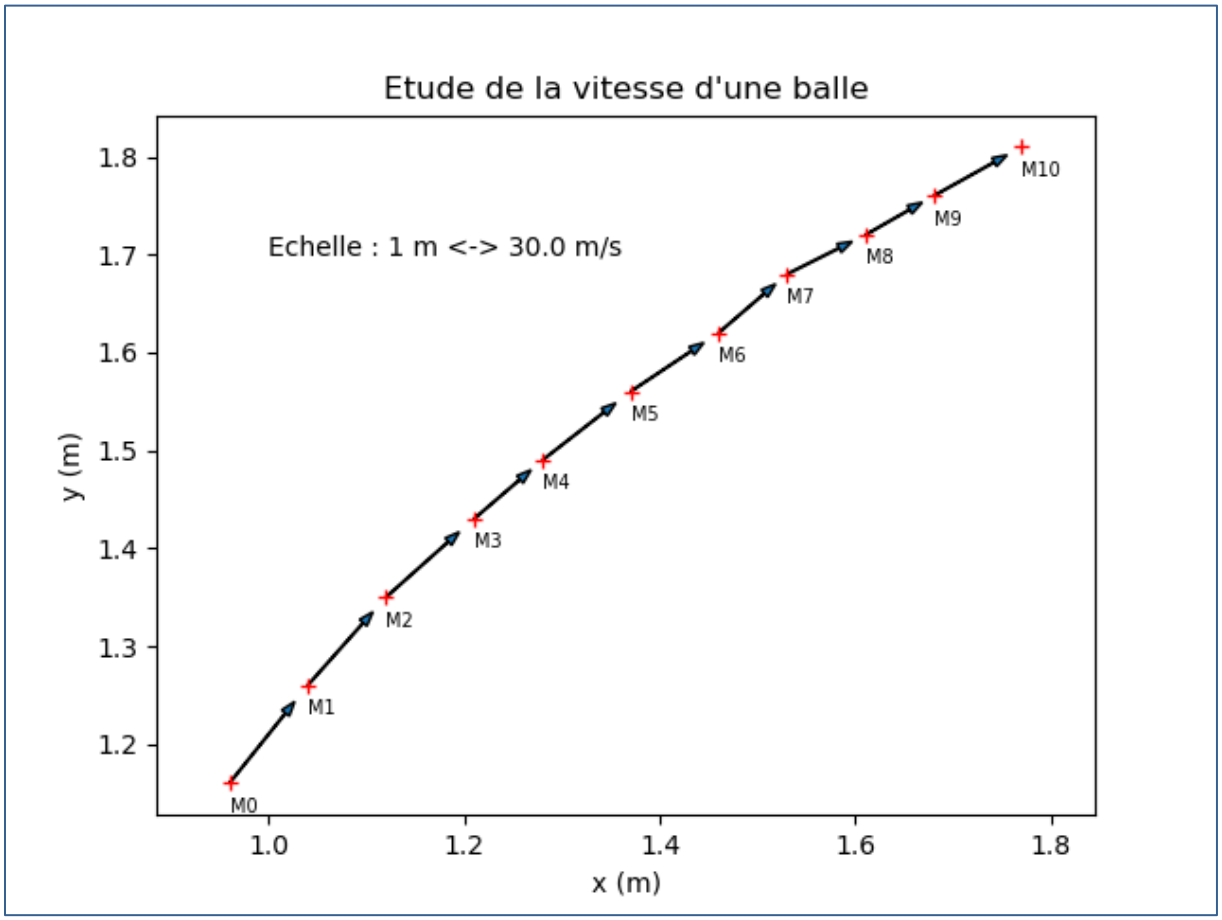
Doc. 20 Ajout de l'échelle et des noms des points (mouvementBalle.py)

```
'''
Fichier mouvementBalle.py
'''
import matplotlib.pyplot as plt

"""
Programme des docs 13 et 19
"""

# Pour nommer les points Mi
for i in range(0,nbrePoints) :
    plt.text(x[i], y[i]-0.03, 'M'+str(i), fontsize=7)
# Pour afficher l'échelle sur le graphe
plt.text(1, 1.7, "Echelle : 1 m <-> "+str(echelle)+" m/s")

plt.plot(x, y, 'r+')
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.title("Etude de la vitesse d'une balle")
plt.axis('equal') # Pour avoir un repère orthonormé
plt.show()
```



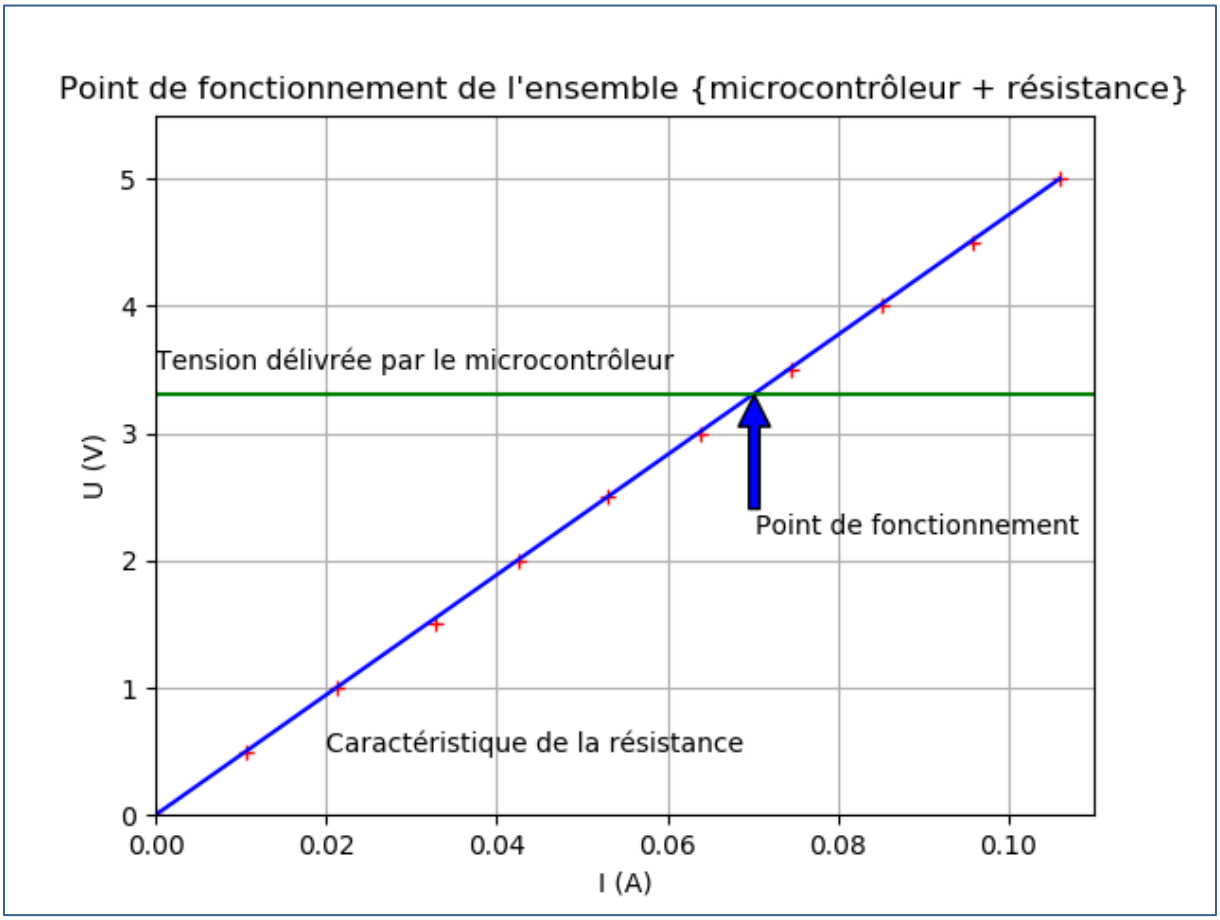
• La méthode *annote()* permet d'ajouter du texte ainsi qu'une flèche pour nommer précisément des courbes ou un point de fonctionnement comme dans l'exemple ci-dessous.

Doc. 21 Compléments au programme du doc 18 (etudeResistance.py)

```python
'''
Fichier etudeResistance.py
'''
import matplotlib.pyplot as plt

list_U_V = [0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5.0]
list_I_A = [0,10.6E-3,21.3E-3,32.9E-3,42.5E-3,53.1E-3,63.8E-3,74.5E-3,85.1E-3,95.7E-3,106E-3]

plt.plot(list_I_A,list_U_V, 'r+')
plt.xlabel("I (A)")
plt.ylabel("U (V)")
plt.title("Etude de la loi d'Ohm - U en fonction de I")
plt.grid(True)
plt.axis([0, 0.11, 0, 5.5])
plt.text(0.02, 0.5, 'Caractéristique de la résistance')
plt.plot([0, 0.106], [0, 5], c = 'blue')
plt.text(0, 3.5, 'Tension délivrée par le microcontrôleur')
plt.plot([0.0, 0.11], [3.3, 3.3], c = 'green')
plt.annotate('Point de fonctionnement', xy=(0.0702, 3.3), xytext=(0.0702, 2.2),arrowprops=dict(facecolor='blue'))
plt.title("Point de fonctionnement de l'ensemble {microcontrôleur + résistance}")
plt.show()
```



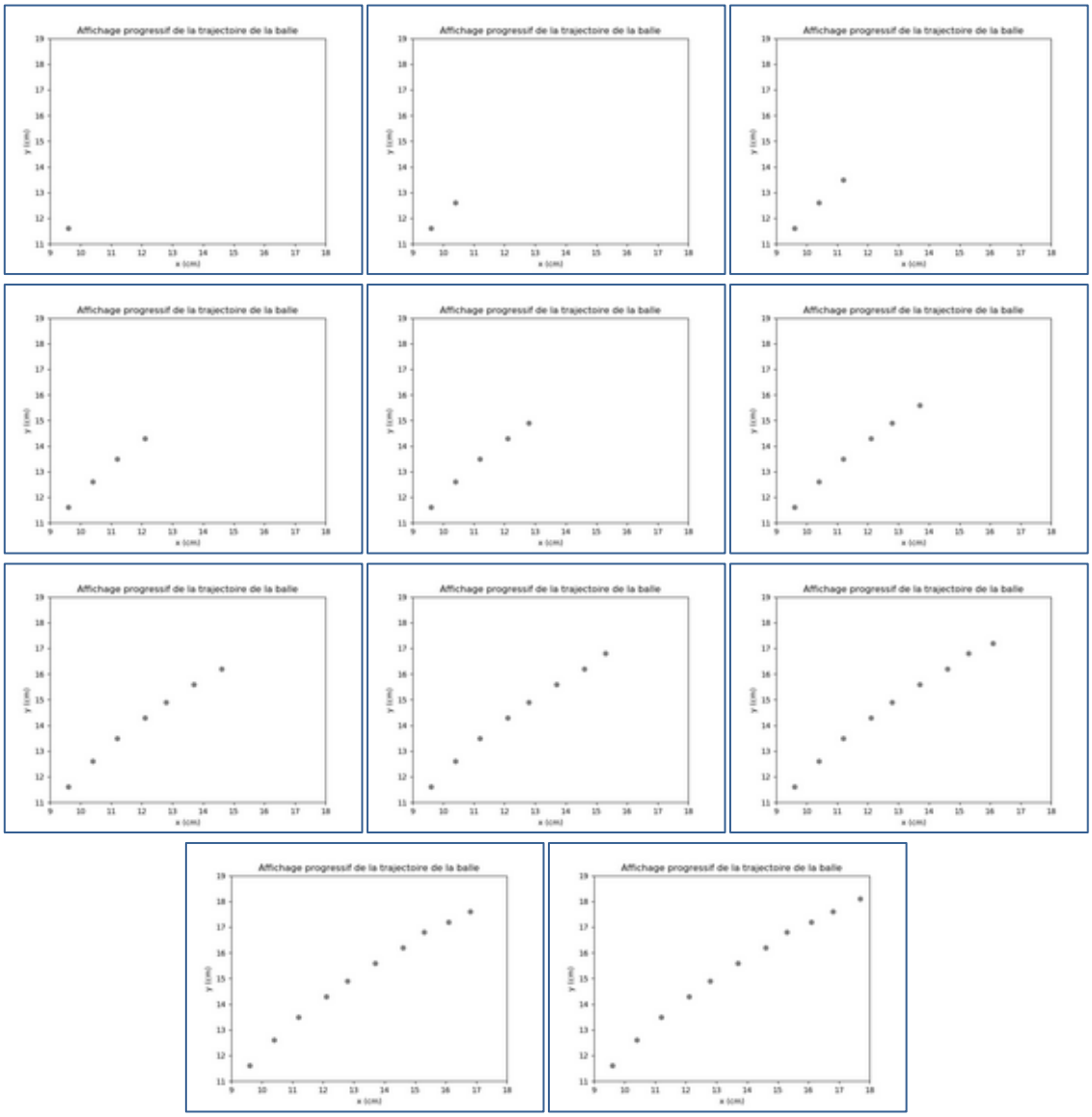
5. Les animations

- La bibliothèque "matplotlib" permet de réaliser des animations en faisant apparaître les points, les courbes ou les vecteurs un à un.
- Elles sont utiles notamment à des fins pédagogiques en permettant, par exemple, d'expliquer les étapes d'une construction.
- Nous l'utilisons également pour créer des animations sur les ondes mécaniques.

Doc. 22 Affichage progressif des points de la trajectoire du doc 12 (animation.py)

```
'''
Fichier animation.py
'''
import matplotlib.pyplot as plt

x = [0.96, 1.04, 1.12, 1.21, 1.28, 1.37, 1.46, 1.53, 1.61, 1.68, 1.77] # en m
y = [1.16, 1.26, 1.35, 1.43, 1.49, 1.56, 1.62, 1.68, 1.72, 1.76, 1.81] # en m
dt = 0.04 # en s
nbrePoints = len(x)
plt.axis([0.9,1.8,1.1,1.9])
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.title("Affichage progressif de la trajectoire de la balle")
for i in range(0, nbrePoints) :
    plt.plot(x[i], y[i], c = 'grey', marker = 'o')
    plt.pause(dt)
plt.show()
```



6. La récupération des données des logiciels de pointage

• Pour éviter de saisir manuellement les coordonnées des points d'une trajectoire, il est possible de récupérer les données d'un logiciel de pointage. Cela nécessite de mettre en œuvre le protocole suivant.

- Exporter les valeurs dans un fichier .csv (possible dans la plupart des logiciels de pointage tels que PyMécaVidéo ou Regressi).
- Ouvrir le fichier avec un tableur, chaque grandeur (t, x, y) correspondant à une colonne.
- Copier les cellules et utiliser le « collage spécial » pour coller les valeurs en les transposant afin que chaque grandeur corresponde à une ligne.
- Supprimer les valeurs en colonne.
- Enregistrer le fichier au format .csv.
- L'ouvrir avec le Bloc-notes ou Notepad++.
- Utiliser l'outil de remplacement pour remplacer les virgules par des points et les points-virgules par des virgules.
- Copier les lignes ainsi obtenues pour les insérer dans votre fichier Python.
- Corriger les noms des variables en début de ligne, ajouter le signe '=' et les crochets [ ] en début et fin de liste.

D. Les autres bibliothèques nécessaires

1. Obtention d'une régression linéaire

• Elle nécessite l'import d'une nouvelle bibliothèque "scipy.stats" et plus spécifiquement du méthode *linregress*.

L'appel de la fonction *linregress(abscisses, ordonnées)* renvoie une liste contenant plusieurs informations dont les 3 premières sont :

- La pente,
- L'ordonnée à l'origine,
- Le coefficient de corrélation.

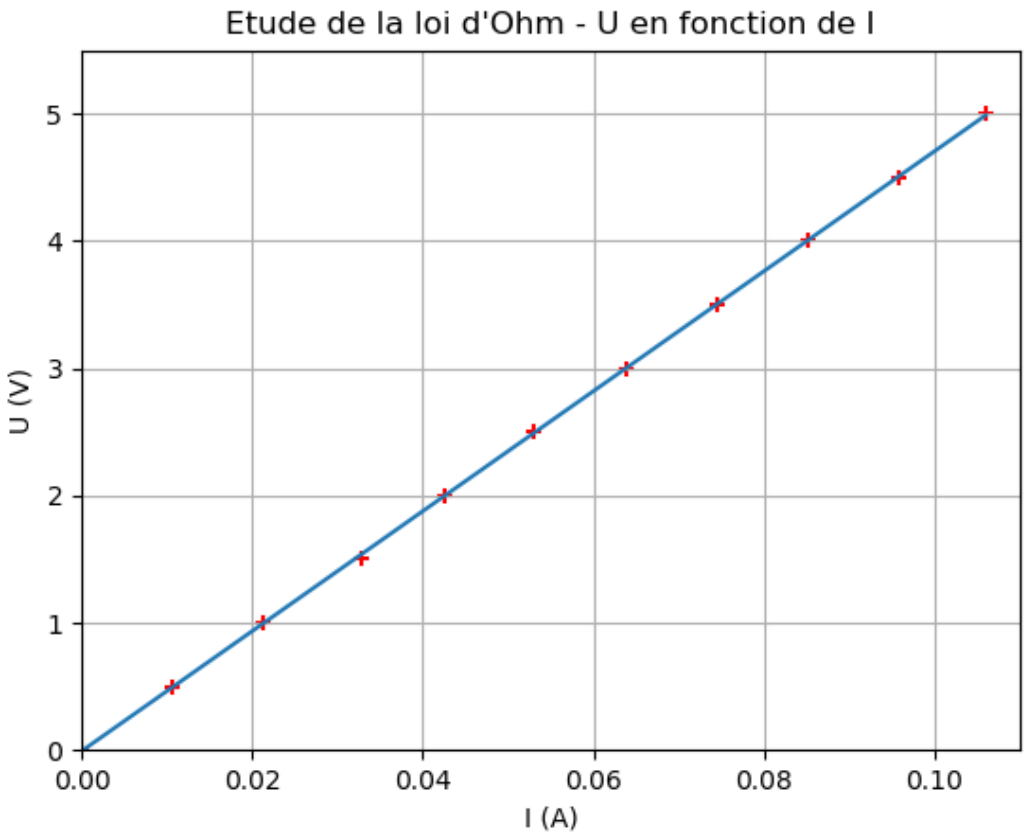
Doc. 23 Exemple pour la caractéristique de la résistance (etudeResistance.py)

```python
'''
Fichier etudeResistance.py
'''
import matplotlib.pyplot as plt
import scipy.stats as st

list_U_V = [0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5.0]
list_I_A = [0,10.6E-3,21.3E-3,32.9E-3,42.5E-3,53.1E-3,63.8E-3,74.5E-3,85.1E-3,95.7E-3,106E-3]

plt.plot(list_I_A,list_U_V, 'r+')
plt.xlabel("I (A)")
plt.ylabel("U (V)")
plt.title("Etude de la loi d'Ohm - U en fonction de I")
plt.grid(True)
plt.axis([0, 0.11, 0, 5.5])
regression = st.linregress(list_I_A,list_U_V)
pente = regression[0]
ordonneeOrigine = regression[1]
coefficientCorrelation = regression[2]
print("La modélisation de U = f(I) a pour équation U =",ordonneeOrigine,"+",pente,"x I")
print("Avec un coefficient de corrélation de",coefficientCorrelation)
# Pour tracer la droite donnée par la régression linéaire
plt.plot([0,max(list_I_A)],[ordonneeOrigine,ordonneeOrigine+pente*max(list_I_A)])
plt.show()
```

La modélisation de U = f(I) a pour équation U = -0.009875484760103692 + 47.15393737380212 x I
Avec un coefficient de corrélation de 0.9999603370231097



2. Récupération des valeurs du microcontrôleur

- Il est possible de récupérer avec Python les valeurs envoyées par le microcontrôleur.
- Pour ce faire, il est nécessaire :
  - D'importer la bibliothèque "serial",
  - De préciser le port visible dans le logiciel Arduino,
  - D'indiquer le même débit de transmission que dans le code Arduino téléversé.

Doc. 24 Instructions spécifiques nécessaires (microcontroleur.py)

```
'''
Fichier microcontroleur.py
'''
import serial as sr

serie = sr.Serial(port='COM5', baudrate=9600)
nombre = float(serie.readline())
"""
Traitement des données
"""
serie.close()
```

3. Fonctions mathématiques

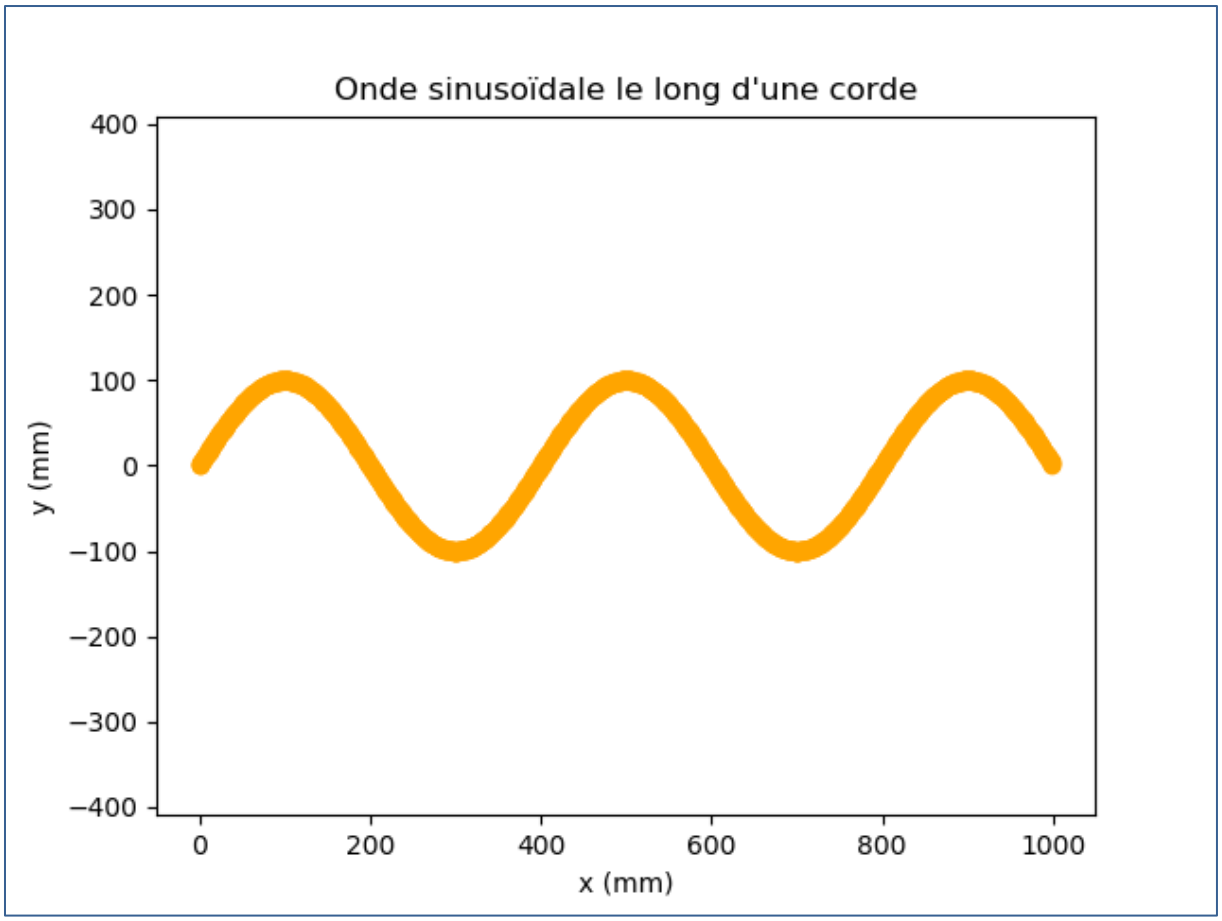
- Pour utiliser le réel π, les fonctions trigonométriques, logarithmiques, … il est nécessaire d'importer la bibliothèque "math".

Doc. 25 Représentation d'une onde sinusoïdale le long d'une corde (corde.py)

```
'''
Fichier corde.py
'''
import matplotlib.pyplot as plt
from math import pi, sin

L = 1000                # en mm
periode = 50            # en ms
longueurOnde = L/2.5
amplitude = 100         # en mm

list_x = []
list_y = []
plt.axis('equal')
plt.xlabel("x (mm)")
plt.ylabel("y (mm)")
plt.title("Onde sinusoïdale le long d'une corde")
for point in range(0,L) :
    list_x.append(point)
    list_y.append(amplitude*sin(2*pi*list_x[point]/longueurOnde))
plt.plot(list_x, list_y, c = 'orange', marker = 'o')
plt.show()
```


Onde sinusoïdale le long d'une corde
400
300
200
100
0
−100
−200
−300
−400
y (mm)
0
200
400
600
800
1000
x (mm)