Python et Arduino pour le physicien

April 4, 2019

1 Ceci est un NoteBook : un fichier au format *.ipynb* (Intercative PYthon NoteBook)

JupyterLab est un éditeur interactif de programmes en langage **Python**. Il a l'intérêt de s'éxécuter par blocs et de pouvoir intercaller des blocs de texte permettant de guider l'utilisateur. C'est l'ensemble de ces blocs qui constitue ce que l'on appelle un **Notebook**.

L'ensemble des blocs de programme qui suivent a été conçu pour que son utilisateur puisse s'approprier de manière autonome et progressive le langage de programmation Python dont l'utilisation est conseillée dans les nouveaux programmes du lycée.

L'ensemble des fonctionnalités présenté n'est évidemment pas exhaustif mais largement suffisant pour répondre à la plupart des mises en uvres envisageables dans les classes de lycée.

---

2 1 - Se familiariser avec l'environnement JupyterLab :

2.1 1.1 - Comprendre les icônes (ci-dessus) :

- Disquette : enregistre le NoteBook ;
- + : crée un nouveau bloc ;
- Ciseaux : supprime un bloc ;
- Double page et presse-papier : copie/colle un bloc ;
- Triangle : valide le bloc ou lit le bloc et passe au bloc suivant (équivalent à shift-enter) ;
- Carré : stoppe l'exécution du bloc ;
- Flèche arrondie : relance le bloc;
- Menu déroulant : permet de préciser le type de contenu du bloc (traitement de texte (**Markdown**) ou **Code**)

2.2 1.2 - Écrire en langage Markdown :

Double-cliquez **ici** pour entrer dans ce bloc de type **Markdown**.

Markdown est un traitement de texte très simplifié qui permet par exemple d'écrire : # des titres ## des sous-titres ### des sous-sous-titres d'écrire en **gras** ou en *italique* de faire des listes : - listes à puces - comme ici

Appuyez maintenant sur les touches shift+Entrer.

Vous avez validé le bloc et pouvez maintenant voir son contenu édité.

Un intérêt du `Markdown`, c'est que les formules peuvent être écrites en LaTeX comme ici : $v_{moy} = \frac{d}{\Delta t} = 7,12 \; m \cdot s^{-1}$

ou là $\boxed{f(n) = \sum_{i=0}^{n} A_i \times \sin(\omega t + \phi_i)}$ et là $\boxed{\sum_{i=1}^{N} \vec{F_i}_{ext} = m \times \vec{g} - \mu \times \vec{v} = \frac{d\vec{p}}{dt} = m \times \vec{a} + \frac{dm}{dt} \times \vec{v}}$

et encore ici $\boxed{\iiint_{sphre} \vec{g(r)} \cdot \vec{dS} = -4\pi \times GM_{intrieure}}$ et là $\boxed{\vec{rot}\vec{B} = \mu_0 \vec{j} + \frac{1}{c^2} \cdot \frac{\partial \vec{E}}{\partial t}}$

Plus d'informations sur Markdown ici.

Plus d'information sur LaTeX là.

2.3 1.3 - Comprendre et manipuler les blocs :

- il faut spécifier pour chaque bloc s'il contient du code Python (Code) ou du texte (Markdown) dans le menu déroulant à droite des icônes au dessus de ce NoteBook ;
- on peut sélectionner un bloc en cliquant dessus : une barre bleue apparait à sa gauche ;
- on peut masquer le bloc en cliquant sur la barre bleue ;
- on peut déplacer un bloc en le sélectionnant par sa gauche et en le glissant ailleurs ;
- on entre dans un bloc par un double-clic ;
- on valide ou on exécute le contenu d'un bloc avec la commande : **shift** + **entrer** comme fait plus haut.

3 2 - "Coder" en Python :

3.1 2.1 - Effectuer des opérations :

3.1.1 a - Effectuer des opérations de base :

```
In [1]: print('bonjour') # Affiche la chaine de caractère : bonjour. NB : tout ce qui est écrit
```

```
bonjour
```

```
In [2]: print(3)  # Affiche l'entier 3
        print(3.) # Affiche le réel 3
```

```
3
3.0
```

```
In [3]: a=3             # Stocke 3 dans la variable a
        print(a)        # Affiche la valeur stockée dans a
        print('a =',a)  # Affiche la chaine de caractère 'a =' puis la valeur stockée dans a
        print(float(a)) # Affiche le contenu de a converti en réel (float)
```

```
3
a = 3
3.0
```

```
In [4]: a=29
        b=7
        c = a + b
        d = a * b
        e = a / b  # division
        f = a // b # division euclidienne
        g = a % b  # reste de la division euclidienne
        h = a**b   # a^b
        print('c =',c,'/ d =',d,'/ e =',e,'/ f =',f,'/ g =',g,'/ h =',h)
```

```
c = 36 / d = 203 / e = 4.142857142857143 / f = 4 / g = 1 / h = 17249876309
```

```
In [5]: a=[None] # Vide la variable a
        print(a)
        a=7
        a=a+1    # Ajoute 1 au contenu stocké dans a puis affiche la nouvelle valeur stockée dan
        print(a) # Affiche le contenu de a
        a+=2     # a devient a + 2
        print(a)
```

```
[None]
8
10
```

```
In [6]: i="Bonjour !"
        j=867
        k=7.19
        l=(3,4)
        m=(l,5,'coucou',3.14)
        print(i)
        print(type(i)) # Affiche à quel type de variable appartient i
        print(j)
        print(type(j)) # Affiche à quel type de variable appartient j
        print(k)
        print(type(k)) # Affiche à quel type de variable appartient k
        print(l)
        print(type(l)) # Affiche à quel type de variable appartient l
        print(m)
        print(type(m)) # Affiche à quel type de variable appartient m
```

```
Bonjour !
<class 'str'>
867
<class 'int'>
7.19
<class 'float'>
(3, 4)
```

```
<class 'tuple'>
((3, 4), 5, 'coucou', 3.14)
<class 'tuple'>
```

```
In [7]: n = 'Bonjour '
        o = 'tout le monde !'
        p = n + o # Ici le signe + correspond à une concaténation, puisque k et l sont des chain
        print(p)
```

```
Bonjour tout le monde !
```

```
In [8]: age = input("Quel est votre âge ?") # Pour plus d'interactivité...
        print(f'Ha bon ! Vous avez',age,'ans ? Vous ne les faites pas !')
```

```
Quel est votre âge ? 23


Ha bon ! Vous avez 23 ans ? Vous ne les faites pas !
```

```
In [9]: print('age*2 =',age*2) # Ici on demande d'afficher age*2 mais age est une chaine de cara
        age=int(age)           # Ici on convertit la chaine de caractères en un entier (int)
        print('age*2 =',age*2) # Ici on demande d'afficher age*2 et age est maintenant un entier
```

```
age*2 = 2323
age*2 = 46
```

Définition d'une fonction nouvelle :

```
In [10]: def v(d,Dt):
             return d / Dt   # définit la fonction v qui calcule le rapport entre d et Dt
         print('v =',v(3,2),'m/s')
```

```
v = 1.5 m/s
```

Boucles conditionnelles :

```
In [11]: i = 0               # i prend la valeur 0
         while i < 5 :       # tant que i est inférieur à 5
             print(i+1)      # affiche i+1
             i+=1            # incrémente i de 1
```

```
1
2
3
4
5
```

```
In [12]: c = 'PHYSIQUE & CHIMIE'
         for i in c :                # pour i appartenant à c
             print(i)                # affiche i
```

```
P
H
Y
S
I
Q
U
E

&

C
H
I
M
I
E
```

```
In [13]: for n in range(1,9) :      # pour n compris dans l'intervalle [1,9]
             print('Si n =',n,'alors 2^n =',2**n)             # affiche 2^n
```

```
Si n = 1 alors 2^n = 2
Si n = 2 alors 2^n = 4
Si n = 3 alors 2^n = 8
Si n = 4 alors 2^n = 16
Si n = 5 alors 2^n = 32
Si n = 6 alors 2^n = 64
Si n = 7 alors 2^n = 128
Si n = 8 alors 2^n = 256
```

Un exemple mathématique amusant bien connu utilisant les boucles conditionnelles est la fameuse conjecture de Syracuse dont la démonstration raportera $1 000 000 $ $ à celui qu'il la démontrera :

Cette conjecture est définie de la façon suivante :

Pour n'importe quel entier N supérieur à 1 : - si N est paire, diviser N par 2; - si N est impaire, multiplier N par 3 et lui ajouter 1.

Essayez, pour voir :

```
In [14]: def syrac(n):                          # on définit la fonction syrac qui reçoit n com
             seq=[n]                            # on place l'entier n dans la liste seq
             while seq[-1] != 1:                # tant que le dernier nombre de la liste est di
                 if seq[-1] % 2 == 0:              # si il est divisible par 2 (pair)
                     seq.append(seq[-1] // 2)         # ajoute à la liste ce nombre divisé par
```

```
            else:                                  # sinon
                seq.append(seq[-1] * 3 + 1)        # ajoute à la liste ce nombre multiplié p
        return seq                                 # renvoit la séquence complété

    N = int(input('choisir un entier supérieur à 1 :'))
    print(syrac(N))
```

```
choisir un entier supérieur à 1 : 4587


[4587, 13762, 6881, 20644, 10322, 5161, 15484, 7742, 3871, 11614, 5807, 17422, 8711, 26134, 1306
```

3.1.2 b - Un exemple de programme avec des boucles conditionnelles en chimie :

```
In [15]: print("Soit l'équation 4 P + 5 O2 -> 2 P2O5")
         nP=input("Quelle est la quantité de matière initiale de phosphore en mole ? : ")
         nO2=input("Quelle est la quantité de matière initiale de dioxygène en mole ? : ")
         print ("\n")
         nP = float(nP)
         nO2 = float(nO2)

         if nP/4 < nO2/5 :
             print ("Tout le phosphore a été consommé.")
             print("La quantité de matière finale de dioxygene est : nf(O2) =", nO2-5*nP/4,"mol"
             print("La quantité de matière finale de pentaoxyde de phosphore est : nf(P2O5) =",

         elif nP/4 > nO2/5:
             print ("Tout le dioxygène a été consommé.")
             print("La quantité de matière finale de phosphore est : nf(P) =", (nP-4*nO2/5),"mol
             print("La quantité de matière finale de pentaoxyde de phosphore est : nf(P2O5) =",(

         else :
             print("Il ne reste ni phosphore, ni dioxygène.")
             print("La quantite de matière finale de pentaoxyde de phosphore est : nf(P2O5) =",2
```

```
Soit l'équation 4 P + 5 O2 -> 2 P2O5


Quelle est la quantité de matière initiale de phosphore en mole ? :  4
Quelle est la quantité de matière initiale de dioxygène en mole ? :  7




Tout le phosphore a été consommé.
La quantité de matière finale de dioxygene est : nf(O2) = 2.0 mol
La quantité de matière finale de pentaoxyde de phosphore est : nf(P2O5) = 2.0 mol
```

3.1.3 c - Effectuer des opérations plus évolués :

Par défaut, Python ne connait pas les fonctions scientifiques telles que sinus, cosinus ou exponentielle,...

Il faut importer au besoin les modules complémentaires. C'est ce que l'on fait ici avec le module **numpy** (https://docs.python.org/fr/)

```
In [16]: import numpy as np # on importe le module numpy qui sera appelé par la commande 'np.'
```

```
In [17]: u = np.pi                                   # pi
         v = np.cos(np.pi/4)                         # cos(pi/4)
         w = np.log(5)                               # ln (5) / log(5) s'écrira np.log10(5)
         x = np.exp(-3/5)                            # exp(-3/5)
         y = np.arccos(np.sqrt(3)/2)*180/np.pi       # arccos(racine(3)/2) converti en degrès
         z = 7*np.array([[1,2,3],[4,5,6],[7,8,9]])          # array = tableau
         print('u =',u)
         print('w =',v)
         print('w =',w)
         print('x =',x)
         print('y =',y)
         print('z =\n',z)
```

```
u = 3.141592653589793
w = 0.7071067811865476
w = 1.6094379124341003
x = 0.5488116360940265
y = 30.000000000000004
z =
 [[ 7 14 21]
 [28 35 42]
 [49 56 63]]
```

Remarques :

- Il est fréquent (et normal) de commettre des erreurs et que le programme ne fonctionne pas. Dans ce cas, un message d'erreur apparait sur fond rouge. C'est en général tout en bas de ce message qu'apparait l'information la plus utile pour résoudre le problème (quelques exemples ci-dessous à corriger).
- En cas de doute sur l'utilisation d'une fonction, il est toujours possible de demander de l'aide en tapant : help(fonction).
- La touche tab permet d'ouvrir une proposition de différentes fonctions commençant par les caractères déjà inscrits.

```
In [18]: a = cosh(5)
         print(a)
```

```
NameError                                 Traceback (most recent call last)

<ipython-input-18-b5a77f7c5ff0> in <module>
----> 1 a = cosh(5)
      2 print(a)


NameError: name 'cosh' is not defined
```

```
In [19]: x = -5
         y = 2*x**2+8*x-10
         z = x/y
```

```
---------------------------------------------------------------------------

ZeroDivisionError                         Traceback (most recent call last)

<ipython-input-19-0e82ca7e33d5> in <module>
      1 x = -5
      2 y = 2*x**2+8*x-10
----> 3 z = x/y


ZeroDivisionError: division by zero
```

```
In [20]: a = (1;2;3;4)
         mean(a)
```

```
  File "<ipython-input-20-a2f0f5b9c84d>", line 1
a = (1;2;3;4)
      ^
SyntaxError: invalid syntax
```

u = 3/2 print(u)

3.2 2.3 - Tracer des graphiques :

Là encore, il va falloir importer un module spécifique pour tracer les graphiques : matplotlib.pyplot (https://www.science-emergence.com/Articles/Tutoriel-Matplotlib/)

```
In [21]: import numpy as np
         import matplotlib.pyplot as plt
```

Les commandes suivantes permettent de tracer la fonction $f(t) = e^{-\frac{t}{\tau}} \times \cos(\omega t)$ sur l'intervalle $[0; 60s]$.

Remarque importante : En informatique, on ne peut tracer qu'un nuage de points en nombre forcément limité. Pour tracer une fonction, on doit donc créer une liste d'abscisses pour un intervalle et un pas de son choix, puis générer une liste d'ordonnées et tracer la seconde liste en fonction de la première.

On définit donc d'abord les bornes et le nombre de points :

```
In [22]: t_min=0.    # Le . permet d'imposer un type réel (float) plutôt qu'un type entier (int)
         t_max=60.
         N=121        # Nombre de valeurs sur l'intervalle
```

En python, la fonction **linspace** du module **numpy** affecte à une variable un tableau (array) de valeurs comprises entre deux valeurs extrêmes et régulièrement réparties dans l'intervalle.

On crée donc la variable t :

```
In [23]: t=np.linspace(t_min,t_max,N)
         print(type(t))
         print(t)        # Remarque : les commandes print() sont là pour illustrer mais n'ont au

<class 'numpy.ndarray'>
[ 0.   0.5  1.   1.5  2.   2.5  3.   3.5  4.   4.5  5.   5.5  6.   6.5
  7.   7.5  8.   8.5  9.   9.5 10.  10.5 11.  11.5 12.  12.5 13.  13.5
 14.  14.5 15.  15.5 16.  16.5 17.  17.5 18.  18.5 19.  19.5 20.  20.5
 21.  21.5 22.  22.5 23.  23.5 24.  24.5 25.  25.5 26.  26.5 27.  27.5
 28.  28.5 29.  29.5 30.  30.5 31.  31.5 32.  32.5 33.  33.5 34.  34.5
 35.  35.5 36.  36.5 37.  37.5 38.  38.5 39.  39.5 40.  40.5 41.  41.5
 42.  42.5 43.  43.5 44.  44.5 45.  45.5 46.  46.5 47.  47.5 48.  48.5
 49.  49.5 50.  50.5 51.  51.5 52.  52.5 53.  53.5 54.  54.5 55.  55.5
 56.  56.5 57.  57.5 58.  58.5 59.  59.5 60. ]
```

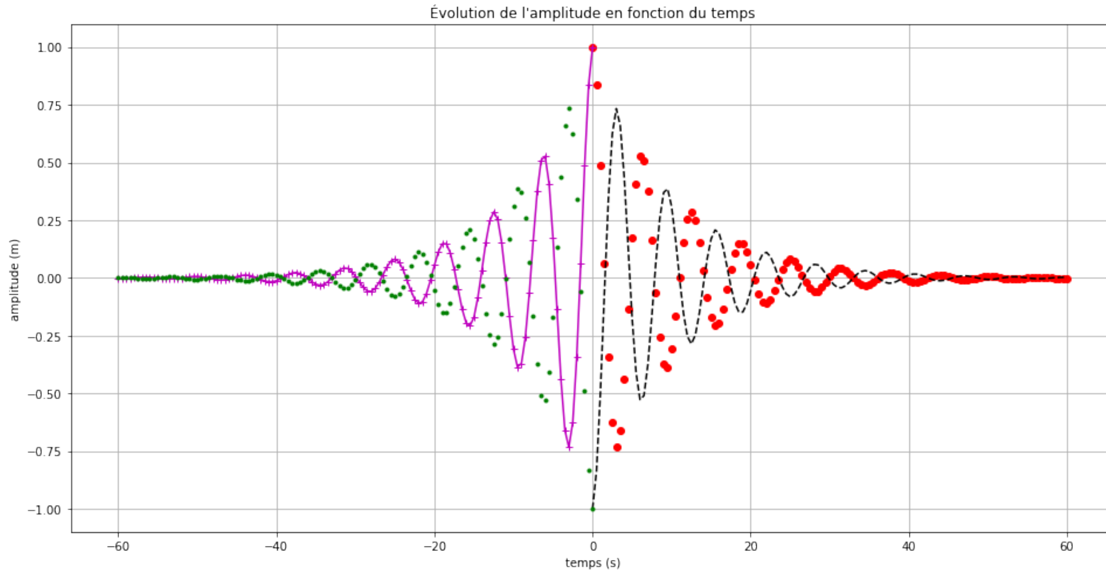
On définit les constantes ω et τ et on calcule la liste f qui est aussi un tableau (array)

```
In [24]: w=1.
         tau=10.
         f=np.exp(-t/tau)*np.cos(w*t)
         print(type(f))
         print(f)        # Remarque : les commandes print() sont là pour illustrer mais n'ont au

<class 'numpy.ndarray'>
[ 1.00000000e+00  8.34782355e-01  4.88885743e-01  6.08840737e-02
 -3.40712213e-01 -6.23931275e-01 -7.33404480e-01 -6.59909874e-01
 -4.38150422e-01 -1.34409336e-01  1.72049812e-01  4.08866892e-01
  5.26952626e-01  5.09823446e-01  3.74376780e-01  1.63738930e-01
 -6.53773795e-02 -2.57308876e-01 -3.70437921e-01 -3.85647380e-01
 -3.08677165e-01 -1.66408322e-01  1.47318689e-03  1.53032057e-01
  2.54163928e-01  2.85873993e-01  2.47308098e-01  1.54227388e-01
  3.37189829e-02 -8.32546875e-02 -1.69509286e-01 -2.07674765e-01
```

```
-1.93348115e-01 -1.34895291e-01 -5.02678083e-02  3.81329478e-02
 1.09149618e-01  1.47728232e-01  1.47879185e-01  1.13223836e-01
 5.52279014e-02 -1.02426082e-02 -6.70729689e-02 -1.02714767e-01
-1.10798818e-01 -9.20456319e-02 -5.34212225e-02 -5.90385602e-03
 3.84806514e-02  6.95879674e-02  8.13628814e-02  7.28747989e-02
 4.80490129e-02  1.42810915e-02 -1.96333384e-02 -4.57011949e-02
-5.85361230e-02 -5.63776941e-02 -4.11605341e-02 -1.77075280e-02
 7.67972748e-03  2.88442359e-02  4.12084136e-02  4.27007692e-02
 3.40047828e-02  1.81359316e-02 -4.89688490e-04 -1.72276361e-02
-2.83195649e-02 -3.16931363e-02 -2.72891400e-02 -1.68838495e-02
-3.49644434e-03  9.43959872e-03  1.89237346e-02  2.30530973e-02
 2.13657346e-02  1.48121715e-02  5.39736262e-03 -4.39136081e-03
-1.22153967e-02 -1.64209277e-02 -1.63628534e-02 -1.24605727e-02
-5.99801052e-03  1.26072576e-03  7.53208759e-03  1.14345558e-02
 1.22754162e-02  1.01481877e-02  5.83580016e-03  5.60786069e-04
-4.34418171e-03 -7.76031271e-03 -9.02556607e-03 -8.04694529e-03
-5.26822599e-03 -1.51446700e-03  2.23838735e-03  5.10747893e-03
 6.50188996e-03  6.23388797e-03  4.52472605e-03  1.91366784e-03
-8.99149142e-04 -3.23274595e-03 -4.58369476e-03 -4.72766552e-03
-3.74564499e-03 -1.97582802e-03  9.04269955e-05  1.93877685e-03
 3.15509168e-03  3.51335189e-03  3.01092332e-03  1.84791907e-03
 3.60824385e-04 -1.06964666e-03 -2.11233172e-03 -2.55881818e-03
-2.36079575e-03]
```

On trace la fonction $f(t)$ et on ajoute un titre, un libellé pour chaque axe et une grille :

```
In [25]: plt.rcParams['figure.figsize'] = [16,8]    # Définit la taille du graphique (largeur x h
         plt.plot(t,f,'ro')   #crée le graphe f(t) sous forme de rond (o) rouges rouge (r)
         plt.plot(-t,f,'m+-')  #crée le graphe f(-t) en trait plein (-) de couleur magenta (m)
         plt.plot(t,-f,'k--') # crée le graphe -f(t) en pointillés (--) noirs (k)
         plt.plot(-t,-f,'g.') # crée le graphe -f(-t) sous formes de croix (+) vertes (g)...
         plt.rcParams['figure.figsize'] = [12,6]    # définit la taille du graphe
         plt.title("Évolution de l'amplitude en fonction du temps")
         plt.xlabel('temps (s)')
         plt.ylabel('amplitude (m)')
         plt.grid()          # Affiche la grille
         plt.show()          # affiche le graphe
```



3.3 2.4 - Traiter des données à partir d'un fichier *.csv* :

Pour traiter des données, le module **pandas** (http://www.python-simple.com/python-pandas/panda-intro.php) permet de lire et de générer des cichier *.csv*.

Ces fichiers peuvent facilement être récupérés à partir de données d'un logiciel de pointage.

```
In [26]: import pandas as pd
```

On se propose ici de traiter un fichier *.csv* obtenu par pointage du centre de gravité d'un ballon de basket lors d'un lancé. Le pointage a été réalisé dans LoggerPro puis exporté (menu Ficher -> Exporter en CSV). Le fichier *Basket.csv* a été placé dans le même dossier que ce fichier

Attention : tous les logiciels ne créent pas les fichiers *.csv* avec le même type de séparateurs. Ici, LoggerPro a utilisé le symbole " , " et il faut le préciser dans Python. Ce pourra être une autre fois un ";", un ".", un " " ou encore une tabulation ""̂

```
In [27]: nom_fichier = 'basket.csv'                  # Attention le fichier .csv doit être placé
         data = pd.read_csv(nom_fichier, sep = ',') # ici on place les données du fichier .csv d
         print(type(data))                          #  un dataframe est un tableau dans lequel t
         data.head()                                # La commande .head() permet de voir les 5 p
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Out[27]:    VideoAnalysis: Time (s)  VideoAnalysis: X (m)  VideoAnalysis: Y (m)  \
         0                    0.180              0.011257             -0.003101
         1                    0.247              0.208170              0.300507
         2                    0.313              0.408141              0.556470
         3                    0.380              0.617850              0.768648
         4                    0.447              0.822571              0.938625
```

```
   VideoAnalysis: X Velocity (m/s)  VideoAnalysis: Y Velocity (m/s)
0                         2.969692                         4.311287
1                         3.009474                         3.977743
2                         3.059634                         3.468555
3                         3.075597                         2.860572
4                         3.061033                         2.219164
```

```
In [28]: t = data['VideoAnalysis: Time (s)']
         x = data['VideoAnalysis: X (m)']     # Ici, on extrait t, x et y du tableau sous la for
         y = data['VideoAnalysis: Y (m)']
         print(type(x))
         print(t)
```

```
<class 'pandas.core.series.Series'>
0     0.180
1     0.247
2     0.313
3     0.380
4     0.447
5     0.514
6     0.580
7     0.647
8     0.714
9     0.781
10    0.847
11    0.914
12    0.981
13    1.047
14    1.114
15    1.181
Name: VideoAnalysis: Time (s), dtype: float64
```

On extrait un élément du dataframe de la manière suivante :

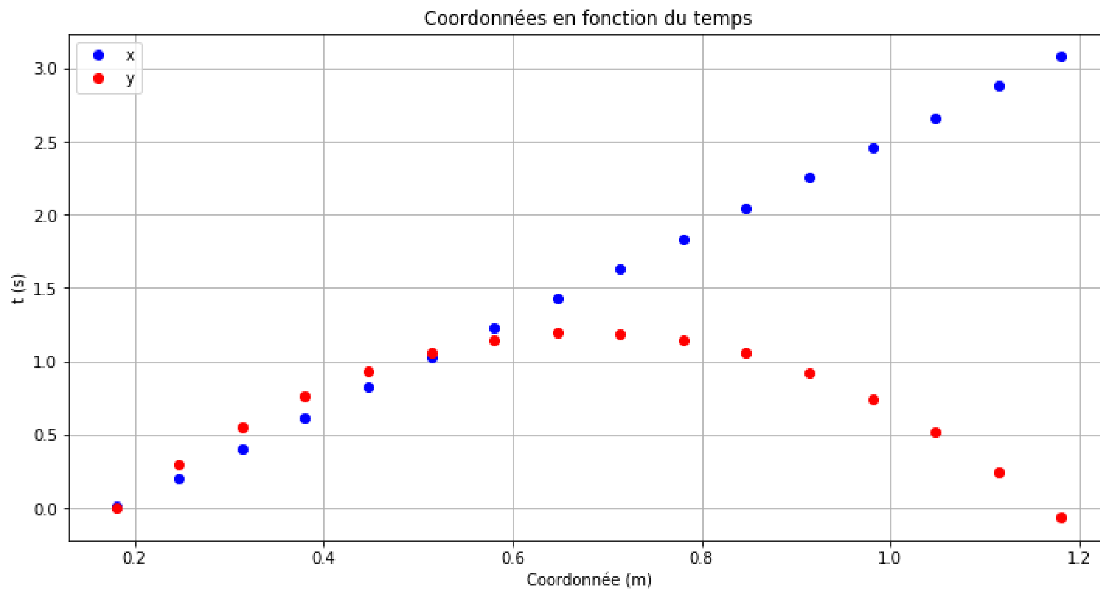
```
In [29]: print(t[0],x[1],y[6]) # affiche le premier élément (0) de la série t, le second (1) de
         print(len(t))         # affiche le nombre d'éléments de la série t
```

```
0.18 0.20816956423099997 1.14965418827
16
```

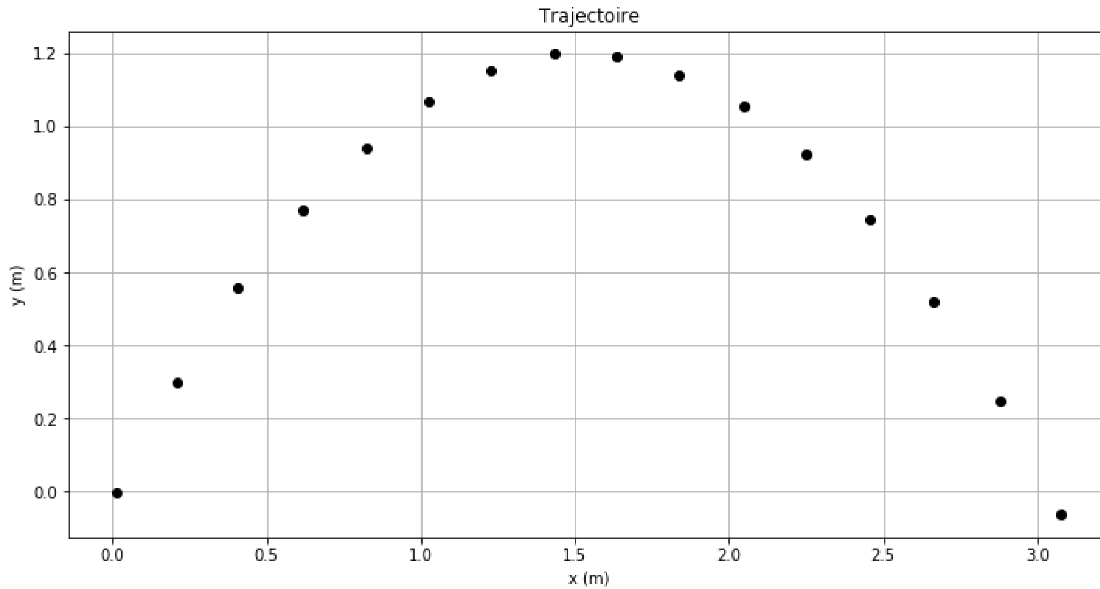
On peut alors tracer n'importe quelle grandeur en fonction d'une autre :

```
In [30]: import matplotlib.pyplot as plt

         plt.rcParams['figure.figsize'] = [12,6]
         plt.plot(t,x,'bo', label='x')
```

```
plt.plot(t,y,'ro', label='y')
plt.legend()
plt.grid()
plt.xlabel("Coordonnée (m)")
plt.ylabel("t (s)")
plt.title("Coordonnées en fonction du temps")
plt.show()
```




```
In [31]: plt.rcParams['figure.figsize'] = [12,6]
         plt.plot(x,y,'ko')
         plt.xlabel("x (m)")
         plt.ylabel("y (m)")
         plt.title("Trajectoire")
         plt.grid()
         plt.show()
         plt.close()
```



3.4 2.5 - Calculer et tracer $v_x(t)$, $v_y(t)$, $a_x(t)$, $a_y(t)$, $E_c(t)$, $E_{pp}(t)$ et $E_m(t)$ (dérivation) :

3.4.1 Remarque importante :

Dans un soucis d'harmonisation avec les mathématiciens, il convient dorénavant d'approximer les vecteurs vitesse et accélération instantannées en un point M_i par les expressions :

$$\boxed{\vec{v}_i = \frac{\overrightarrow{M_i M_{i+1}}}{t_{i+1} - t_i}} \qquad et \qquad \boxed{\vec{a}_i = \frac{\vec{v_{i+1}} - \vec{v}_i}{t_{i+1} - t_i}}$$

```
In [32]: liste_pos = list(zip(x,y)) # la fonction zip transforme 2 Series x et y en 1 liste de t
         liste_temps = list(t)      # On convertit ici la Series t en liste

In [33]: import time                                  # on importe ici ce module uniquement p
                                                      # en créant un délais dans le skript...

         def deriv(Xi,Xf,ti,tf):
             xi, yi = Xi                              # On définit ici une nouvelle fonction
             xf, yf = Xf                              # Elle reçoit les coordonnées de 2 poin
             D_x = xf - xi                            #  et 2 dates  sous la forme de 2 float
             D_y = yf - yi                            #   et renvoit le tuple (vx,vy).
             D_t = tf - ti                            # NB : aucun calcul n'est effectué ici.
             return D_x / D_t, D_y / D_t

         n = len(liste_pos)-1                         # on définit les points de la liste pou
                                                      # n = nb. de positions -1 : on exclue l


         liste_vit = []                               # on crée une liste vide pour accueilli
```

```
    for i in range(0,n):                               # pour i variant de 0 à n
        Xi = (liste_pos[i][0],liste_pos[i][1])         # on extrait le tuple (xi,yi) de la lis
        Xf = (liste_pos[i+1][0],liste_pos[i+1][1])     # on extrait le tuple (xi+1,yi+1) de la
        ti = liste_temps[i]                            # on extrait la date ti de la liste des
        tf = liste_temps[i+1]                          # on extrait la date ti+1 de la liste d
        liste_vit += [deriv(Xi,Xf,ti,tf)]              # on ajoute le tuple (vxi,vyi) à la lis
        #time.sleep(1.0)
        #print(liste_vit)                              # retirer éventuellement les # pour com

    liste_temps2 = liste_temps[0:-1]                   # on crée une nouvelle liste des dates
                                                       #  en excluant la dernière valeur

    n = len(liste_vit)-1                               # n = nb. de positions -1 car pas de ca

    liste_acc = []                                     # on crée une liste vide pour accueilli

    for i in range(0,n):                               # pour i variant de 0 à n
        Vi = (liste_vit[i][0],liste_vit[i][1])         # on extrait le tuple (vxi,vyi) de la l
        Vf = (liste_vit[i+1][0],liste_vit[i+1][1])     # on extrait le tuple (vxi+1,vyi+1) de
        ti = liste_temps[i]                            # on extrait la date ti de la liste des
        tf = liste_temps[i+1]                          # on extrait la date ti+1 de la liste d
        liste_acc += [deriv(Vi,Vf,ti,tf)]              # on ajoute le tuple (axi,ayi) à la lis
```

```
In [34]: data = list(zip(liste_temps[0:-2],liste_pos[0:-2],liste_vit[0:-1], liste_acc))
         # on crée une liste : (t,(x,z),(vx,vz),(ax,az)) uniquement pour les dates où toutes les
         # [N:-M] permet d'exclure les N premières valeurs et les M dernières

         #for i in range (0,vx):
         #    print(data[i])                   # cette boucle permet de voir le fonctionnement du
         #    time.sleep(1.0)
         #print('et ainsi de suite...')
```

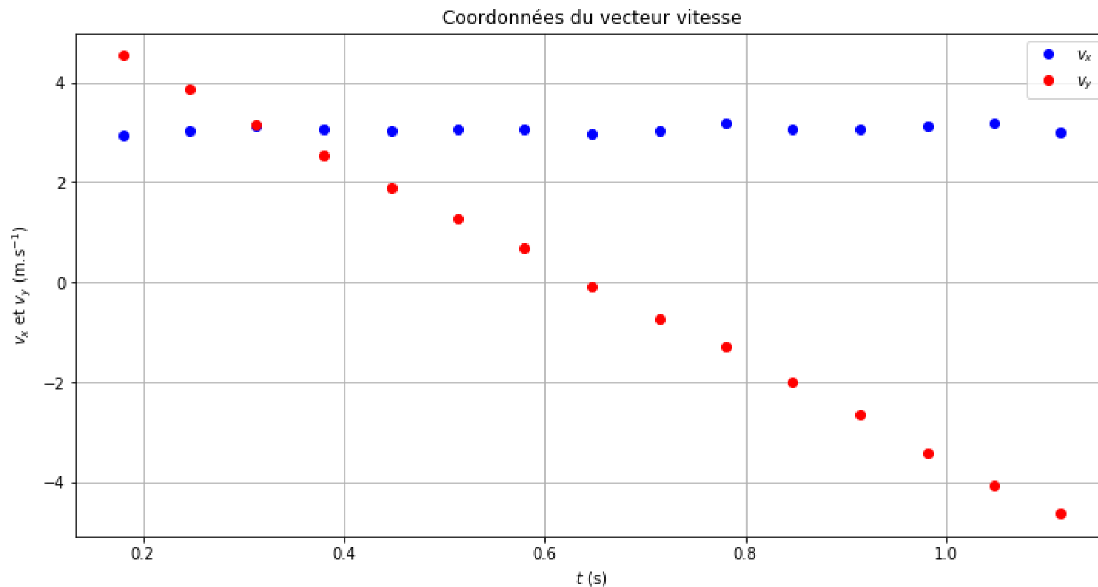
On va à présent récupérer les coordonnées vx et vy qui sont dans liste_vit. Voici la première ligne de cette liste :

```
In [35]: print(liste_vit[0])    # affiche la première ligne de la liste des (vx,vy)

(2.9390007769865667, 4.531459232572687)
```

```
In [36]: vx = [v[0] for v in liste_vit]  # on définit la liste vx extraite à partir de la colonn
         vy = [v[1] for v in liste_vit]  # on définit la liste vy extraite à partir de la colonn
         print('On obtient ainsi : vx =',vx[0],'m/s et vy =',vy[0],'m/s séparément.')

On obtient ainsi : vx = 2.9390007769865667 m/s et vy = 4.531459232572687 m/s séparément.
```

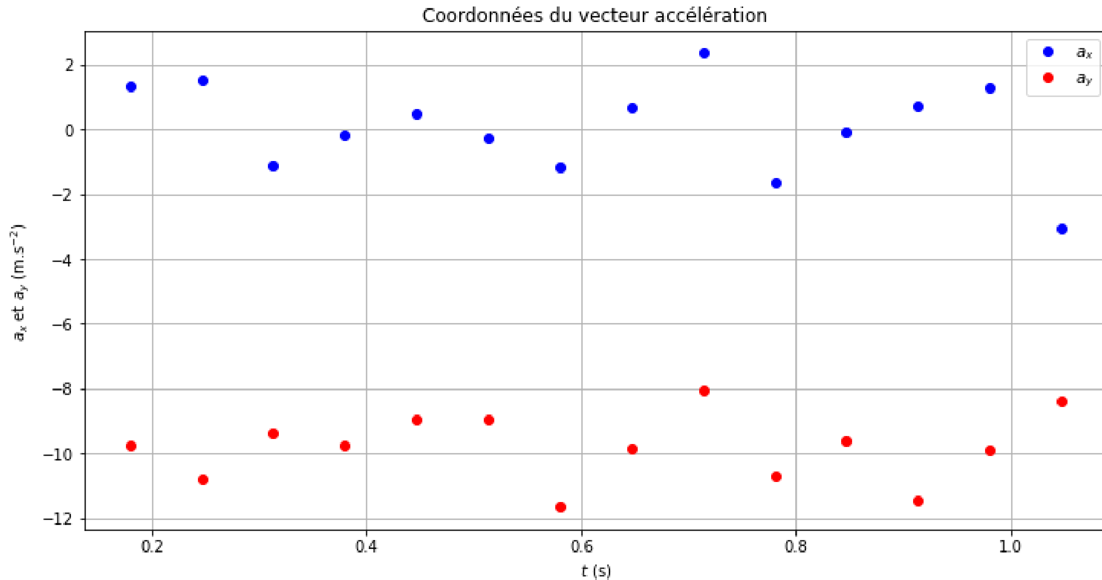
On peut maintenant tracer vx(t) et vy(t) :

```
In [37]: plt.rcParams['figure.figsize'] = [12,6]
         plt.xlabel("$t$ (s)")
         plt.ylabel("$v_x$ et $v_y$ (m.s$^{-1}$)")
         plt.title("Coordonnées du vecteur vitesse")
         plt.grid()
         plt.plot(liste_temps[0:-1], vx, 'ob', label='$v_x$')
         plt.plot(liste_temps[0:-1], vy, 'or', label='$v_y$')
         plt.legend()
         plt.show()
```



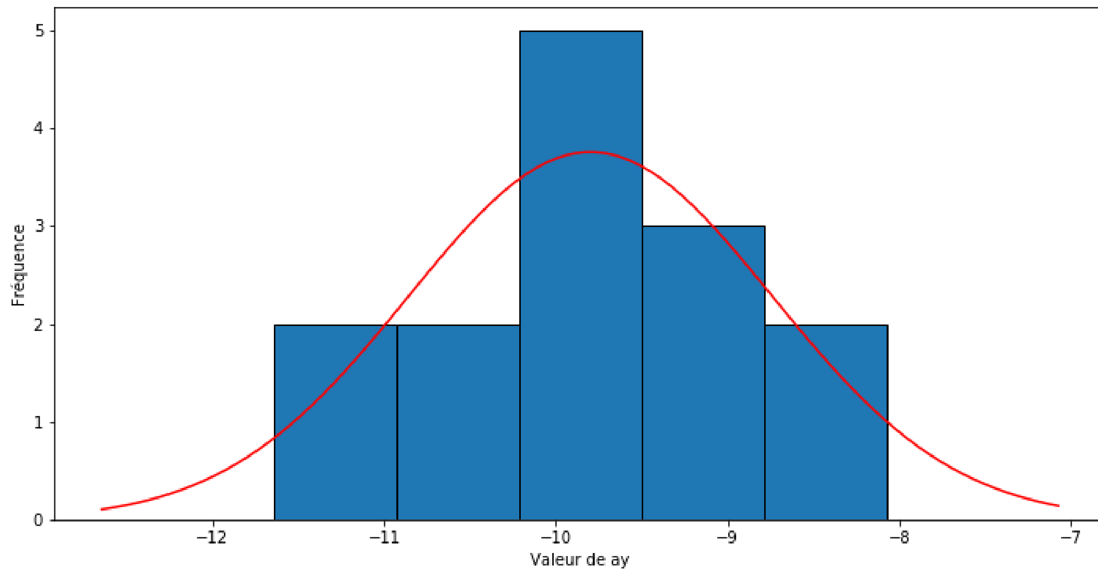
On peut maintenant répéter l'opération pour obtenir les coordonnées du vecteur accélération :

```
In [38]: ax = [a[0] for a in liste_acc]   # on définit la liste ax extraite à partir de la colonn
         ay = [a[1] for a in liste_acc]   # on définit la liste ay extraite à partir de la colonn
         plt.rcParams['figure.figsize'] = [12,6]
         plt.xlabel("$t$ (s)")
         plt.ylabel("$a_x$ et $a_y$ (m.s$^{-2}$)")
         plt.title("Coordonnées du vecteur accélération")
         plt.grid()
         plt.plot(liste_temps2[0:-1], ax, 'ob', label='$a_x$')
         plt.plot(liste_temps2[0:-1], ay, 'or', label='$a_y$')
         plt.legend()
         plt.show()
```

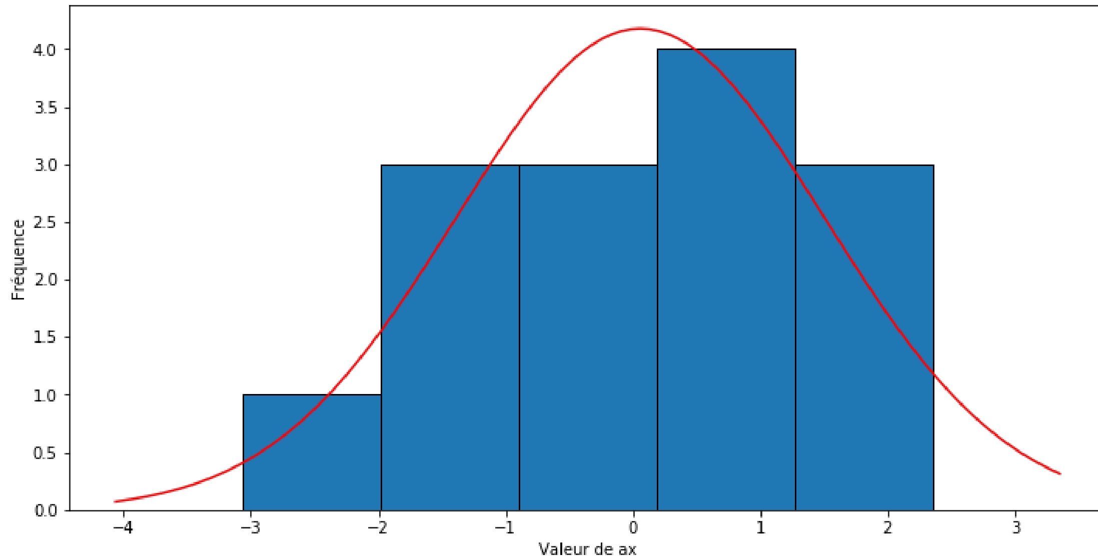


Étant donnée la dispersion (attendues après 2 dérivations successives) des valeurs de ax et ay, on peut envisager une étude statistique :

```
In [39]: import numpy as np
         from scipy import std , mean

         moyenne = mean(ay)
         std_m = std (ay , ddof=1)

         def gauss(x, A, moyenne, ecarttype):
             return A/(ecarttype*np.sqrt(2*np.pi))*np.exp(-(x-moyenne)**2/(2*ecarttype**2))
                                                               # on définit d'abord la f
         N_classes = 5                                         # on demande le nombre de
         A = len(ay) * (max(ay)-min(ay))/N_classes             # on calcule l'air sous l
         x_fit = np.linspace(min(ay)-1,max(ay)+1,100)          # on génère la liste des
         y_fit = gauss(x_fit, A, moyenne, std_m)
         plt.rcParams['figure.figsize'] = [12,6]               # définit la taille du gr
         plt.hist(ay,range=[min(ay),max(ay)],bins = N_classes, edgecolor='k')
         plt.xlabel ("Valeur de ay")
         plt.ylabel ("Fréquence")
         plt.plot(x_fit,y_fit,'r')
         plt.show()
```

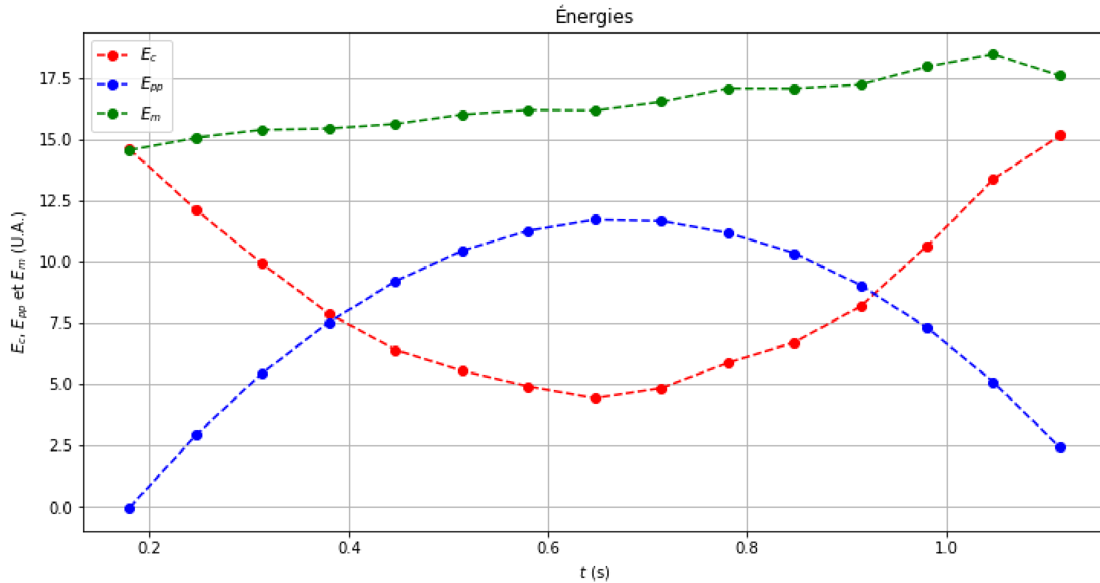



```
In [40]: from scipy import std , mean
         moyenne = mean(ax)
         std_m = std (ax , ddof=1)

         def gauss(x, A, moyenne, ecarttype):
             return A/(ecarttype*np.sqrt(2*np.pi))*np.exp(-(x-moyenne)**2/(2*ecarttype**2))
                                                                # on définit d'abord la f
         N_classes = 5                                          # on demande le nombre de
         A = len(ax) * (max(ax)-min(ax))/N_classes              # on calcule l'air sous l
         x_fit2 = np.linspace(min(ax)-1,max(ax)+1,100)          # on génère la liste des
         y_fit2 = gauss(x_fit2, A, moyenne, std_m)
         plt.rcParams['figure.figsize'] = [12,6]                # définit la taille du gr
         plt.hist(ax,range=[min(ax),max(ax)],bins = N_classes, edgecolor='k')
         plt.xlabel ("Valeur de ax")
         plt.ylabel ("Fréquence")
         plt.plot(x_fit2,y_fit2,'r')
         plt.show()
```



On peut également tracer les énergies en fonction du temps en UA (on ne connait pas la masse ici) :

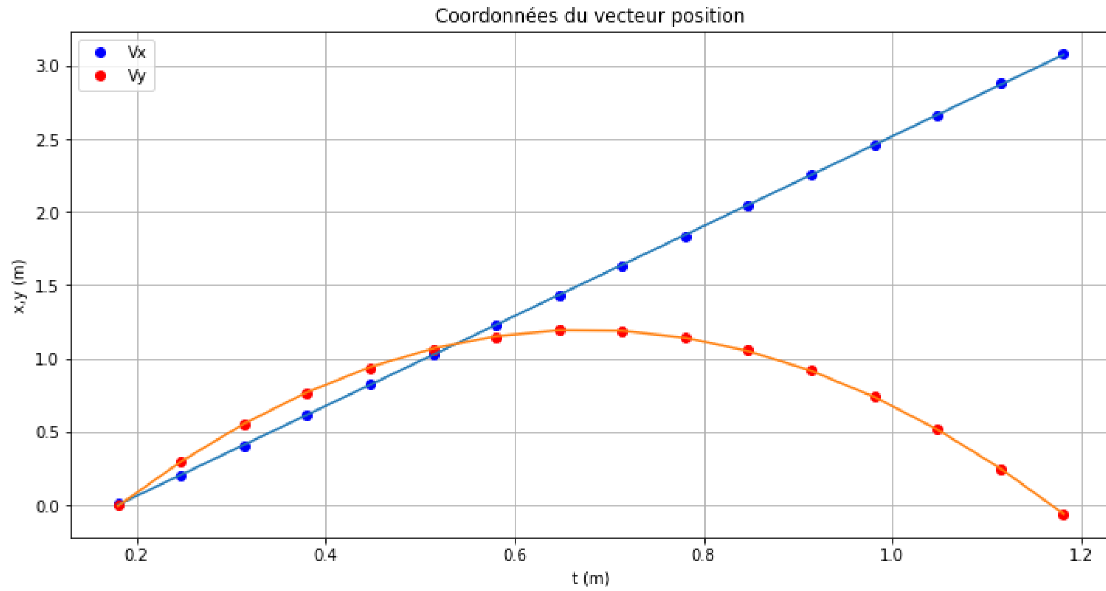
```
In [41]: Ec = []
         Epp = []
         Em = []
         n = len(liste_temps[0:-1])
         for i in range(0,n):
             Eci = (vx[i]**2+vy[i]**2)/2
             Ec.append(Eci)
             Epi = 9.8 * y[i]
             Epp.append(Epi)
             Emi = Eci + Epi
             Em.append(Emi)

         plt.rcParams['figure.figsize'] = [12,6]
         plt.xlabel("$t$ (s)")
         plt.ylabel("$E_c$, $E_{pp}$ et $E_m$ (U.A.)")
         plt.title("Énergies")
         plt.grid()
         plt.plot(liste_temps[0:-1], Ec, 'or--', label='$E_c$')
         plt.plot(liste_temps[0:-1], Epp, 'ob--', label='$E_{pp}$')
         plt.plot(liste_temps[0:-1], Em, 'og--', label='$E_m$')
         plt.legend()
         plt.show()
```



3.5 2.6 - Effectuer une régression polynomiale :

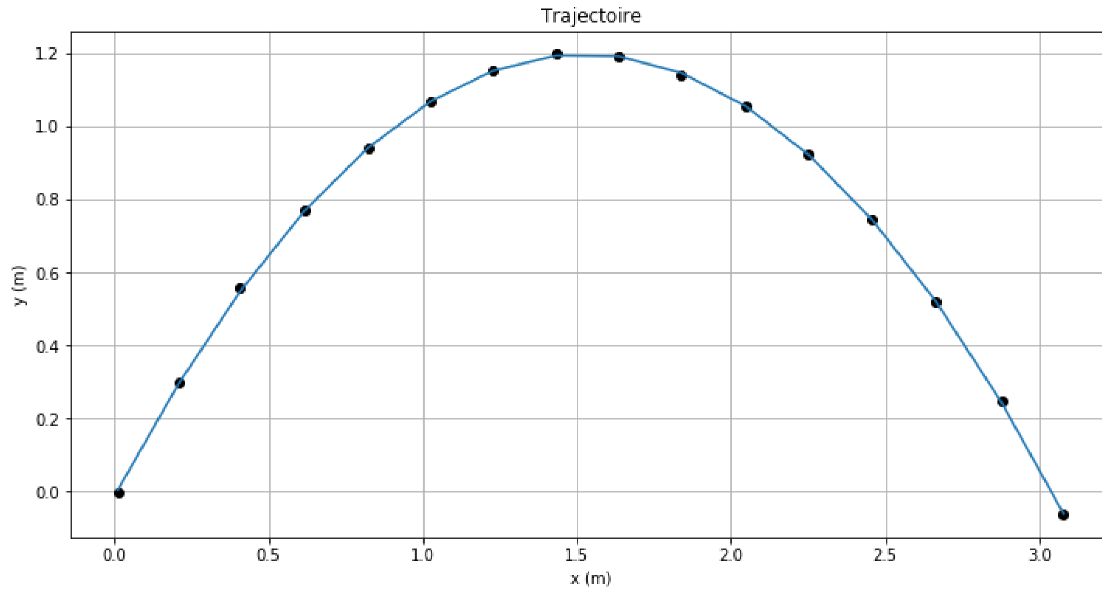
On montre ici comment il est possible de modéliser les fonctions $x(t)$, $y(t)$, $y(x)$, $v_x(t)$, $v_y(t)$, $a_x(t)$ et $a_y(t)$ par une fonction polynomiale à l'aide de la fonction polyfit du module numpy.

```
In [42]: plt.plot(t,x,'bo',label='Vx')
         plt.plot(t,y,'ro',label='Vy')
         plt.legend()
         plt.grid()
         xt_fit = np.polyfit(t,x,1)
         yt_fit = np.polyfit(t,y,2)
         m = xt_fit[0]
         n = xt_fit[1]
         p = yt_fit[0]
         q = yt_fit[1]
         r = yt_fit[2]
         xt_mod = m * t + n
         yt_mod = p * t**2 + q * t + r
         plt.rcParams['figure.figsize'] = [12,6]
         plt.xlabel("t (m)")
         plt.ylabel("x,y (m)")
         plt.title("Coordonnées du vecteur position")
         plt.plot(t,xt_mod)
         plt.plot(t,yt_mod)
         plt.show()
         print('x(t) = ',m,'t +',n)
         print('y(t) = ',p,'t^2 +',q,'t +',r)
```




```
x(t) =  3.06738706225512 t + -0.549568185717162
y(t) =  -4.905648469978689 t^2 + 6.623310272554282 t + -1.0381824933826955
```

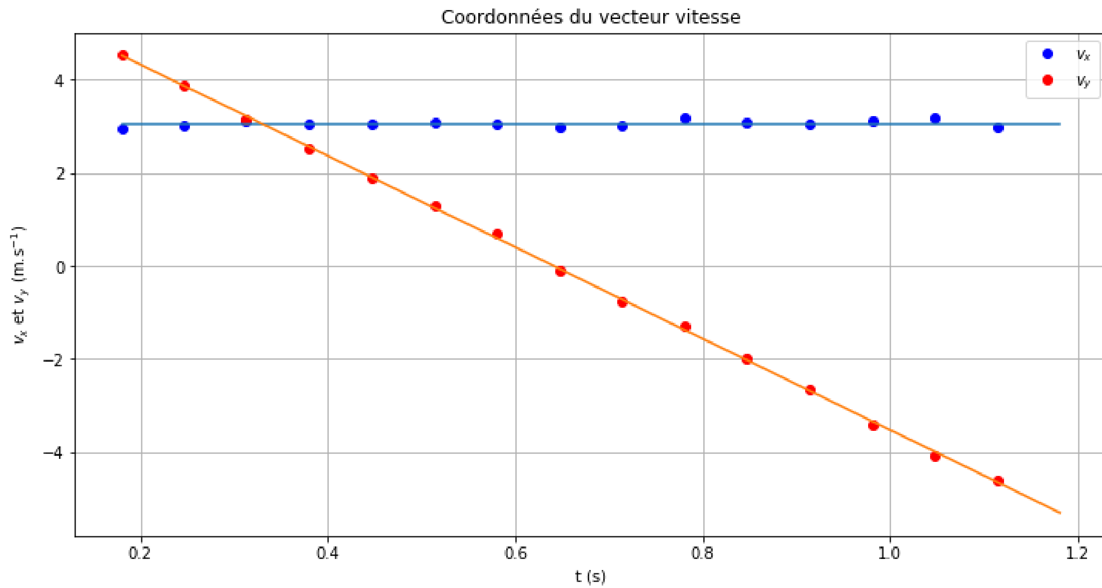
```
In [43]: plt.plot(x,y,'ko')
         plt.xlabel("x (m)")
         plt.ylabel("y (m)")
         plt.title("Trajectoire")
         plt.grid()
         yx_fit = np.polyfit(x,y,2)
         a = yx_fit[0]
         b = yx_fit[1]
         c = yx_fit[2]
         y_mod = a * x**2 + b * x + c
         plt.rcParams['figure.figsize'] = [12,6]
         plt.plot(x,y_mod)
         plt.show()
         print('La trajectoire a pour équation : y =',a,'x^2 +',b,'x +',c)
```



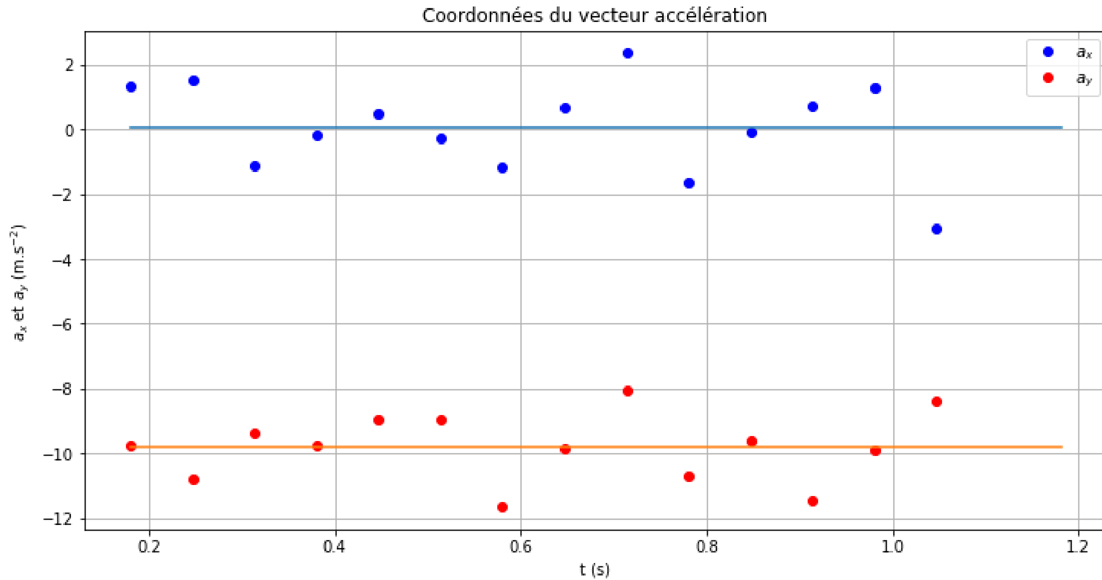

```
La trajectoire a pour équation : y = -0.5204378484234075 x^2 + 1.5865281273677545 x + -0.0125770
```

```
In [44]: plt.plot(liste_temps[0:-1], vx, 'ob', label='$v_x$')
         plt.plot(liste_temps[0:-1], vy, 'or', label='$v_y$')
         plt.legend()
         vxt_fit = np.polyfit(liste_temps[0:-1], vx,0)
         vyt_fit = np.polyfit(liste_temps[0:-1], vy,1)
         d = vxt_fit[0]
         e = vyt_fit[0]
         f = vyt_fit[1]

         print('vx(t) =',d ,' m/s')
         print('vy(t) =',e,'t +',f)

         vxt_mod = d*t/t
         vyt_mod = e * t + f
         t_mod=np.linspace(min(t),max(t),len(vxt_mod))

         plt.rcParams['figure.figsize'] = [12,6]
         plt.grid()
         plt.xlabel("t (s)")
         plt.ylabel('$v_x$ et $v_y$ (m.s$^{-1}$)')
         plt.title("Coordonnées du vecteur vitesse")
         plt.plot(t_mod,vxt_mod)
         plt.plot(t_mod,vyt_mod)
         plt.show()
```

```
vx(t) = 3.0635162807183876  m/s
vy(t) = -9.82095142888766 t + 6.297143725982249
```



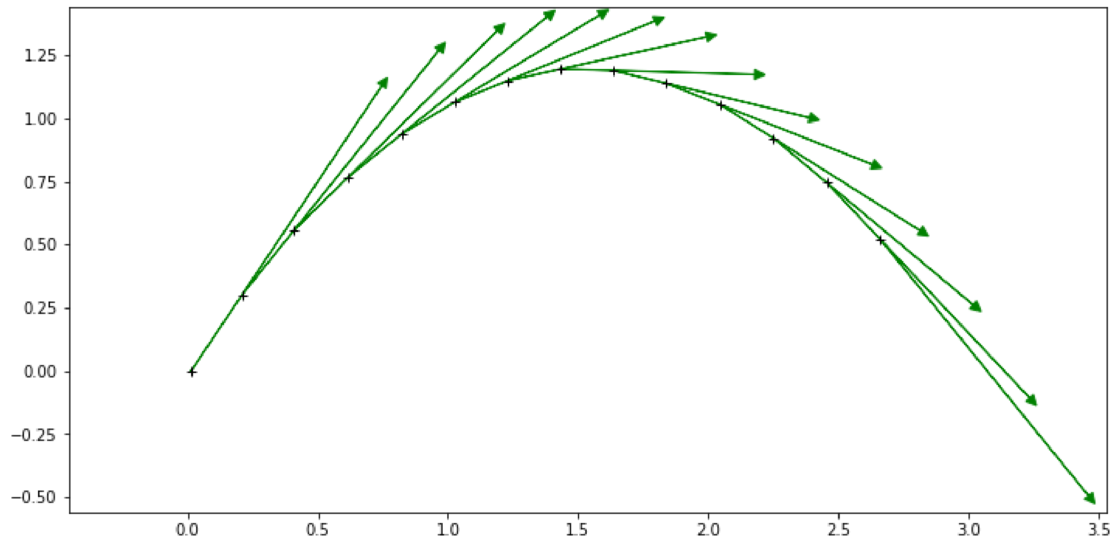

```
In [45]: plt.plot(liste_temps2[0:-1], ax, 'ob', label='$a_x$')
         plt.plot(liste_temps2[0:-1], ay, 'or', label='$a_y$')
         plt.legend()

         h = np.mean(ax)
         i = np.mean(ay)

         axt_mod = h * t/t
         ayt_mod = i * t/t
         t_mod=np.linspace(min(t),max(t),len(axt_mod))

         plt.rcParams['figure.figsize'] = [12,6]
         plt.grid()
         plt.xlabel("t (s)")
         plt.ylabel('$a_x$ et $a_y$ (m.s$^{-2}$)')
         plt.title("Coordonnées du vecteur accélération")
         plt.plot(t_mod,axt_mod)
         plt.plot(t_mod,ayt_mod)
         plt.show()

         print('ax(t) =',h ,' m/s-2')
         print('ay(t) =',i ,' m/s-2')
```



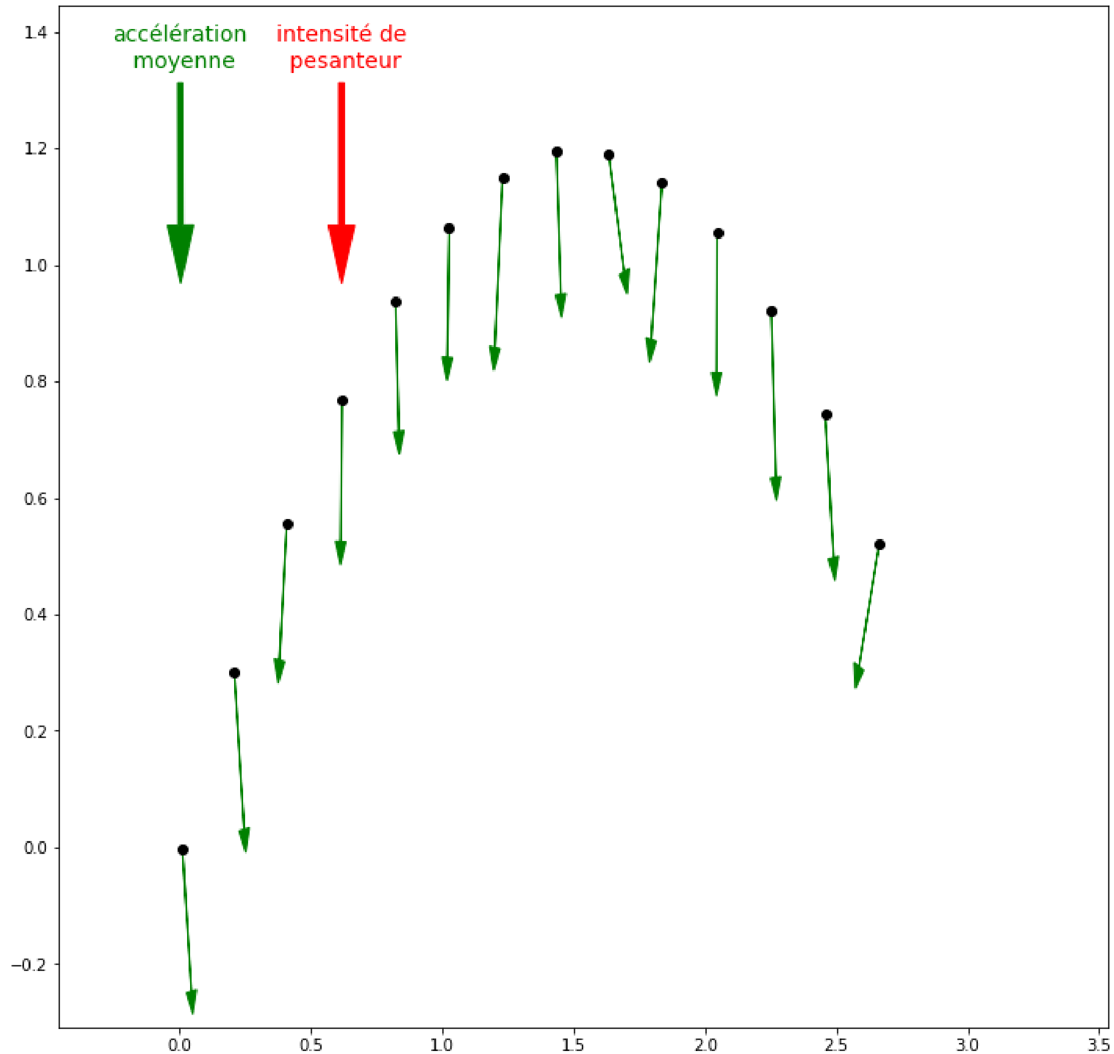

```
ax(t) = 0.057859972360419745  m/s-2
ay(t) = -9.79881984751401  m/s-2
```

3.6 2.7 - Tracer les vecteurs vitesse et accélération :

In [46]:
```
k = 0.25 # on change l'échelle pour que les flèches rentrent dans la figure

for p in data:
    t, (x, y), (vx, vy), (ax, ay) = p
    plt.arrow(x,y,k*vx,k*vy,fc="g",ec="g",head_width=0.04,head_length=0.04)
    plt.plot(x,y,'k+')

# on affine ensuite les limites du cadre pour s'assurer que les vecteurs restent dans c

cx = 0.15
cy = 0.2
x_min=min(xt_mod)
x_max=max(xt_mod)
y_min=min(yt_mod)
y_max=max(yt_mod)

plt.rcParams['figure.figsize'] = [12,12]
plt.xlim(x_min-cx*(x_max-x_min),x_max+cx*(x_max-x_min))
plt.ylim(y_min-2*cy*(y_max-y_min),y_max+cy*(y_max-y_min))
plt.show()
```



In [47]:
```
k = 0.025
ax_moy=0
ay_moy=0
for p in data:
    t, (x, y), (vx, vy), (ax, ay) = p
    plt.arrow(x,y,k*ax,k*ay,fc="g",ec="g",head_width=0.04,head_length=0.04)
    plt.plot(x,y,'ko')
    ax_moy+=ax
    ay_moy+=ay

ax_moy=ax_moy/len(data)
ay_moy=ay_moy/len(data)
plt.arrow(x_min,y_max*1.1,k*ax_moy,k*ay_moy,fc="g",ec="g",head_width=0.1,head_length=0.
plt.text(x_min,y_max*1.02*1.1,'accélération\n moyenne',fontsize=14,color='g',horizontal
plt.arrow(x_min+(x_max-x_min)/5,y_max*1.1,0,-9.81*k,fc="r",ec="r",head_width=0.1,head_l
plt.text(x_min+(x_max-x_min)/5,y_max*1.02*1.1,'intensité de\n pesanteur',fontsize=14,co

plt.rcParams['figure.figsize'] = [12,12]
plt.xlim(x_min-cx*(x_max-x_min),x_max+cx*(x_max-x_min))
plt.ylim(y_min-cy*(y_max-y_min),y_max+cy*(y_max-y_min))
plt.show()
```



3.7 2.8 - Créer un fichier au format *.csv* contenant des données calculées avec Python :

On créer d'abord le dataframe dans lequel on va enregistrer les données :

```
In [48]: nom_fichier = 'basket.csv'
         data = pd.read_csv(nom_fichier, sep = ',')
         t = data['VideoAnalysis: Time (s)']

         newdata = pd.DataFrame({
             't (s)' : t[0:15],
             'Ec (J)' : Ec[0:15],
             'Epp (J)' : Epp[0:15],
             'Em (J)' : Em[0:15]
         })
         newdata.head()
```

```
Out[48]:    t (s)      Ec (J)   Epp (J)     Em (J)
         0  0.180  14.585924 -0.030391  14.555533
         1  0.247  12.110371  2.944965  15.055336
         2  0.313   9.912872  5.453403  15.366274
         3  0.380   7.886264  7.532752  15.419015
         4  0.447   6.407797  9.198526  15.606323
```

La fonction *to_csv* permet de générer le fichier *.csv* qui apparait dans la liste de gauche après exécution du script. - *sep=';'* : indique que les informations sont séparées par un point-virgule; - *index=False* : précise qu'aucun indice de colonne doit être enregistré dans le fichier; - *encoding='utf-8'* : stipule que l'encodage des données dans le fichier est *utf-8*

```
In [49]: nom_fichier = "Energies.csv"
         newdata.to_csv(nom_fichier, sep=';', index=False, encoding='utf-8')
```

4 3 - Communiquer avec une carte Arduino :

4.1 3.1 - Récupérer les données issues du microcontrôleur :

Les données issues de la carte Arduino peuvent être récupérées et traitées avec Python.

Il faut au préalable installer le module **Serial** (https://riptutorial.com/fr/python/topic/5744/communicatio serie-python–pyserial-) qui permet de gérer les ports série : pour cela, il faut retourner dans **Anaconda-Navigator**, puis dans **Environments** (en haut à gauche), choisir "not installed" et rechercher **pyserial**.

Une fois l'installation terminée, il faut importer le module **Serial** et le module **time** :

```
In [ ]: import serial as sr
        import time
```

Il faut ensuite préciser à Python : - le port utilisé et le taux de transfert : le port est indiqué en bas à droite de la fenêtre **Arduino** ; - le taux de transfert choisi et indiqué dans le programme.

5 TRÈS IMPORTANT :

Le moniteur série d'Arduino doit toujours être fermé avant d'activer le port dans Python : la carte Arduino ne peut pas communiquer simultanément avec le moniteur série Arduino et avec Python.

De même le port série doit être désactivé par la commande *port_serie.close()* dans Python si on veut visualiser le moniteur série !

```
In [ ]: # Ouverture du port :
        port_serie = sr.Serial(port = "/dev/cu.usbmodem14301", baudrate = "9600") # adapter la c
```

```
In [ ]: # Fermeture du port :
        port_serie.close()
```

On va ensuite lire les informations qui sont envoyées par la carte sous la forme d'une chaine de caractères.

Le montage est celui d'une résistance de 10 $k\Omega$ en série avec une photo résistance, l'ensemble étant alimenté par une tension de $5V$ et $EA0$ étant pris entre la résistance et la photorésistance.

Ici on utilise un programme Arduino qui affiche **chaque 10 ms** le temps écoulé en millisecondes, une tabulation et la valeur de la tension U mesurée sur l'entrée $EA0$:

```
void setup() {
Serial.begin(9600);        // précise le taux de transfert au bauds
}
void loop() {
Serial.print(millis());    // Affiche le temps écoulé en millisecondes
Serial.print("\t ");       // Affiche une tabulation
float U;                   // Définit la variable réelle U
U = analogRead(0)*5./1023; // Calcule U à partir du mot numérique reçu par EA0, la carte utilisa
Serial.println(U);         // Affiche U et revient à la ligne
delay(10);                 // Attend 10 ms
}
```

```
In [ ]: port_serie = sr.Serial(port = "/dev/cu.usbmodem14301", baudrate = "9600")
        print(port_serie.readline()) # affiche la ligne lue jusqu'au retour à la ligne (println)
        # \r = retour chariot
        # \n = retour à la ligne
        # entre les '' = string
        port_serie.close()
```

On va ensuite récupérer les données envoyées par l'Arduino et les stocker dans une liste à l'aide d'une boucle itérative :

```
In [ ]: port_serie = sr.Serial(port = "/dev/cu.usbmodem14301", baudrate = "9600")
        liste_U = []                         # génère une liste nommée liste_U
        for i in range(5):                   # pour i de 0 à 4 (5 premières lignes en partant de
            mesure = port_serie.readline()   # on crée la variable "mesure" qui prend la valeur d
            liste_U.append(mesure)           # on ajoute la valeur "mesure" à la liste grâce à la
            print(mesure.split())            # la fonction split permet de découper la chaine de
        port_serie.close()                   # ferme le port série
```

On peut maintenant automatiser l'ensemble pour récupérer à caque instant t la mesure de U :

NB : Tant que les crochets à gauche de ce cadre contiennent une *, c'est que l'aquisition n'est pas terminée.

```
In [ ]: port_serie = sr.Serial(port = "/dev/cu.usbmodem14301", baudrate = "9600")
        port_serie.setDTR(False)                # Procédure non indispensable
        time.sleep(0.1)                         # de
        port_serie.setDTR(True)                 # réinitialisation
        port_serie.flushInput()                 # du port

        mesures = []                            # génère une liste nommée mesures
        temps = []                              # génère une liste nommée temps

        N = int(input('Nombre de mesures souhaité :'))
```

```
print('Patientez...')

for i in range(N):                     # prends les N premières valeurs
    val = port_serie.readline().split() # place dans la variable "val" les lignes déconc
    try:                               # permet de sauter à la valeur suivante en cas d
        t = float(val[0])              # t prend la valeur de la colonne
        U = float(val[1])
        mesures.append(U)              # on ajoute U à la liste des mesures
        temps.append(t)                # et t à la liste des temps
    except:
        pass                           # sauf en cas d'erreur
print('Acquisition terminée')
port_serie.close()                     # on ferme le port série
```

Remarque : le même script peut être légèrement adapté pour demander la durée d'aquisition plutôt que le nombre de points de mesure :

```
In [ ]: port_serie = sr.Serial(port = "/dev/cu.usbmodem14301", baudrate = "9600")
        port_serie.setDTR(False)               # Procédure non indispensable
        time.sleep(0.1)                        # de
        port_serie.setDTR(True)                # réinitialisation
        port_serie.flushInput()                # du port

        mesures = []                           # génère une liste nommée mesures
        temps = []                             # génère une liste nommée temps
        duree = int(input('Durée d\'acquisition souhaitée en secondes :'))
        duree = duree * 1000
        end = False

        print('Patientez...')

        while end == False or temps[-1] - temps[0] <= duree:
            val = port_serie.readline().split() # place dans la variable "val" les lignes déconc
            try:                               # permet de sauter à la valeur suivante en cas d
                t = float(val[0])              # attention la première colonne est la colonne 0
                U = float(val[1])
                mesures.append(U)              # on ajoute U à la liste des mesures
                temps.append(t)                # et t à la liste des temps
                end = True
            except:
                pass                           # sauf en cas d'erreur
        print('Acquisition terminée')
        port_serie.close()
```

5.1 3.2 - Effectuer un traitement statistique des données :

On peut analyser la variabilité des mesures de la tension en traçant un histogramme d'abord sur l'ensemble de la plage de mesure (0 à 5 V) pour vérifier que les mesures sont cohérentes :

```
In [ ]: import matplotlib.pyplot as plt

        N = len(temps)
        plt.rcParams['figure.figsize'] = [12,6]
        N_classes=int(N/10)
        plt.hist(mesures,range=[0,5],bins=N_classes, edgecolor='k')
        plt.xlabel("Valeur de U")
        plt.ylabel ("Fréquence")
        plt.show()
```

On peut à présent limiter l'étendue de l'histogramme aux valeurs mesurées, calculer la moyenne et l'écart-type, puis comparer à une loi normale.

Il faut pour cela chargée les fonctions **mean** et **std_m** du module skipy.

```
In [ ]: from scipy import std , mean
        moyenne = mean(mesures)
        std_m = std (mesures , ddof=1)
        print ( "Moyenne numérique = " , moyenne , "\nÉcart type numérique = " , std_m )
```

```
In [ ]: import numpy as np

        def gauss(x, A, moyenne, ecarttype):
            return A/(ecarttype*np.sqrt(2*np.pi))*np.exp(-(x-moyenne)**2/(2*ecarttype**2))
                                                                    # on définit d'abord la fo
        N_classes = int(input('Entrer le nombre de clases'))        # on demande le nombre de
        A = N * (max(mesures)-min(mesures))/N_classes               # on calcule l'air sous la
        x = np.linspace(int(min(mesures)*1000-1)/1000, int((max(mesures))*1000+1)/1000,100)   #
        y = gauss(x, A, moyenne, std_m)
        plt.rcParams['figure.figsize'] = [12,6]                     # définit la taille du gra
        plt.hist(mesures,range=[min(mesures),max(mesures)],bins = N_classes, edgecolor='k')
        plt.xlabel ("Valeur numérique")
        plt.ylabel ("Fréquence")
        plt.plot(x,y,'r')
        plt.show()
```

5.2 3.3 - Construire un graphe pas à pas :

5.2.1 a. Tracer d'un graphe point par point

Cette fonctionnalité nécessite d'importer la fonction **display** du module **IPython** :

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import time
        from IPython import display
```

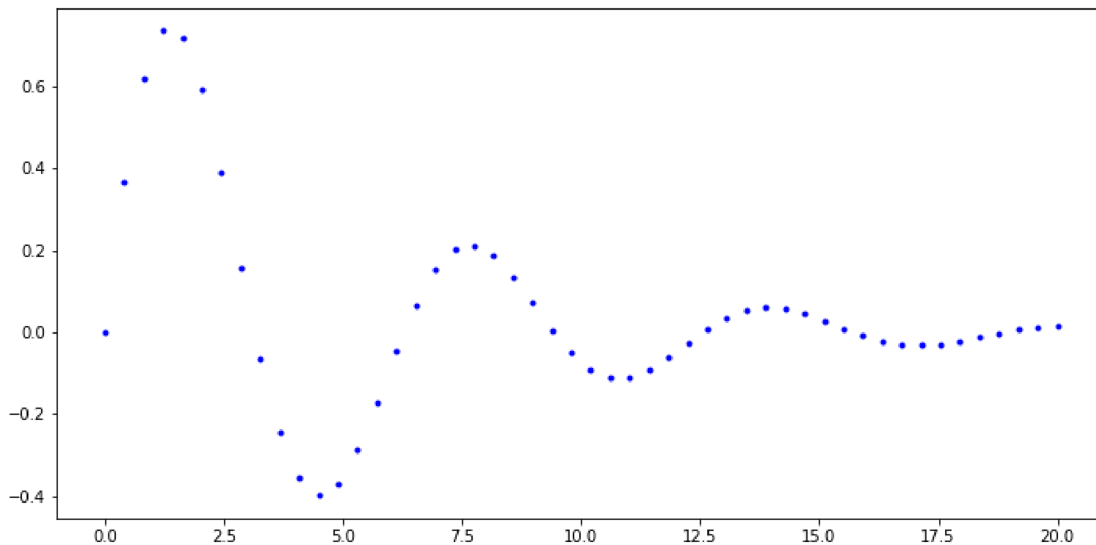
Les deux instructions suivantes permettent d'activer l'affichage du graphique.

```
In [2]: display.display(plt.gcf())
        display.clear_output(wait=True)
```

```
<Figure size 432x288 with 0 Axes>
```

La fonction *draw()* du module **numpy** permet de réaliser le rafraichissement du graphique.
On se propose par exemple de faire apparaitre point par point la fonction $u(t) = sin(t) \times e^{-t/5}$:

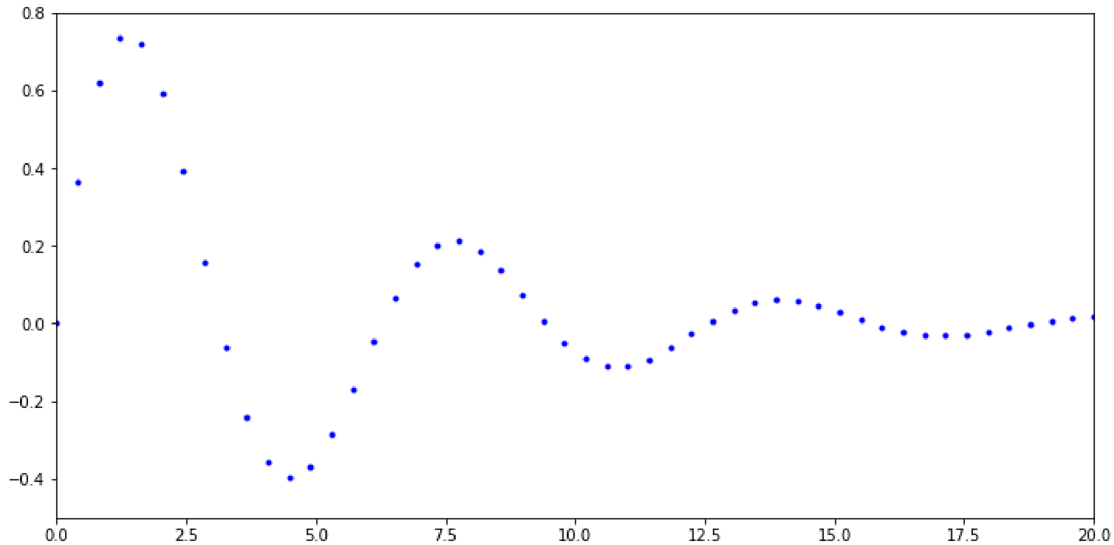
```
In [3]: tmin = 0
        tmax = 20
        N = 50
        t = np.linspace(tmin,tmax,N)

        for i in range(N):
            u = np.sin(t[i]) * np.exp(-t[i]/5)
            plt.rcParams['figure.figsize'] = [12,6]
            plt.plot(t[i], u, 'b.')
            plt.draw()
            display.display(plt.gcf())
            display.clear_output(wait=True)
            time.sleep(0.01) # une pause de 0,01 s
```



On observe que l'échelle s'adapte progressivement à l'espace occupé par la courbe, mais on peut également fixer les limites au préalable :

```
In [4]: tmin = 0
        tmax = 20
        N = 50
        t = np.linspace(tmin,tmax,N)

        for i in range(N):
```

```
    u = np.sin(t[i]) * np.exp(-t[i]/5)
    plt.rcParams['figure.figsize'] = [12,6]
    plt.plot(t[i], u, 'b.')
    plt.xlim(0,20)
    plt.ylim(-0.5,0.8)
    plt.draw()
    display.display(plt.gcf())
    display.clear_output(wait=True)
    time.sleep(0.01) # une pause de 0,01 s
```



La simulation ci-dessous illustre par exemple le concept de modèle en physique et sa limite.

Des élèves du projet "Ambition Sciences" du lycée Sud-Médoc ont établi l'équation de la trajectoire d'un kleenex supposé ponctuel, lancé depuis une hauteur h, avec une vitesse initiale $\vec{v}_0$, suivant un angle α par rapport à l'horizontale et soumis à une force de frottement modélisée par une force $\vec{f} = -\mu \times \vec{v}$. Ils ont obtenu l'équation suivante :

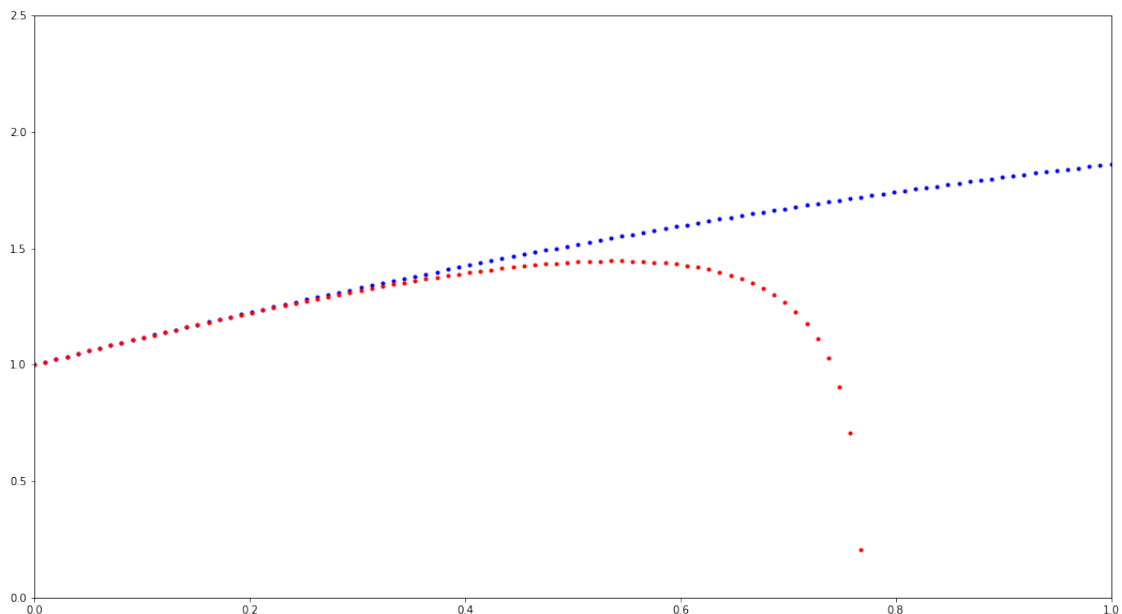
$$y_{frott.} = g \cdot \left(\frac{m}{\mu}\right)^2 \cdot ln\left(1 - \frac{\mu \cdot x}{m \cdot v_0 \cdot cos\alpha}\right) + \left(tan\alpha + \frac{m \cdot g}{\mu \cdot v_0 \cdot cos\alpha}\right) \cdot x$$

On compare ici la trajectoire obtenue à celle où les frottements sont négligés (programme de terminale) :

$$y = -\frac{g}{2 \cdot v_0^2 \cdot cos^2\alpha} \cdot x^2 + tan\alpha \cdot x + h$$

```
In [11]: xmin = 0.
         xmax = 1.0
         Nx = 100
         x = np.linspace(xmin,xmax,Nx)
         g = 9.81
         m = 0.01
         mu =0.05
         a = 50.
```

```
v0 = 6

for i in range(Nx):
    z_sans = - g/2 * x**2 / (v0 * np.cos(a * np.pi / 180))**2 + np.tan(a * np.pi / 180)
    z_avec = g * (m/mu)**2 * np.log (1 - x / (m/mu * v0 * np.cos(a * np.pi / 180))) + (
    plt.rcParams['figure.figsize'] = [18,10]
    plt.plot(x[i],z_sans[i], 'b.')
    plt.plot(x[i],z_avec[i], 'r.')
    plt.xlim(0,1.0)
    plt.ylim(0,2.5)
    plt.draw()
    display.display(plt.gcf())
    display.clear_output(wait=True)
    time.sleep(0.02) # une pause de 0,02 s
```



5.2.2 b. Acquérir point par point des données issues d'un Arduino :

Nous proposons ici de suivre par exemple l'évolution de la température T, mesurée en temps réel par un capteur LM35DZ.

Nous définissons d'abord une fonction que nous nommerons par exemple *graphe* qui reçoit l'abscisse x, l'ordonnée y et la durée Δt,puis place le point (x, y) et temporise pendant la durée Δt :

```
In [ ]: def graphe(x, y, Dt):
            plt.plot(x, y, 'b.')
            plt.draw()
            display.display(plt.gcf())
```

```
            display.clear_output(wait=True)
            time.sleep(Dt)
```

Attention : Le bloc suivant ne pourra s'éxécuter qu'après avoir au préalable adapté les valeurs soulignées en # à son propore dispositif et après avoir branché la carte Arduino sur laquelle on aura téléversé le programme de mesure de la température rappelé ci-dessous :

```
void setup()
{
  Serial.begin(9600);           // Initialise port série et définit le taux de transfert
  analogReference(INTERNAL);    // Améliore la précision de la mesure en réduisant la plage de me
}
void loop()
{
  int Num;                      // définit une variable Num comme un entier
  float T;                      // définit la variable T comme un réel
  float t = millis() / 1000.;   // définit la variable t comme un réel égal au nombre de seconds
  Num = analogRead(0);          // affecte à Num la valeur reçue par l'entrée EA0
  T = Num * 1.1 / 1023 * 100;   // calcule T en řC sachant que 1 V correspond à 100 řC et que le
  Serial.print(t);              // affiche t
  Serial.print("\t ");          // affiche une tabulation
  Serial.println(T);            // affiche T
  delay(1000);                  // attend 1000 ms
}
```

```
In [ ]: import serial as sr

        port_serie = sr.Serial(port = "/dev/cu.usbmodem14301", baudrate = "9600")
        port_serie.setDTR(False)           ################             ######
        time.sleep(0.1)
        port_serie.setDTR(True)
        port_serie.flushInput()

        temps = []
        temperature = []
        duree = int(input('Durée souhaitée en secondes :'))
        end   = False

        while end == False or (temps[-1] - temps[0] < duree):
            val = port_serie.readline().split()
            try:
                t  = float(val[0])
                T  = float(val[1])
                temps.append(t)
                temperature.append(T)
                end = True
                plt.rcParams['figure.figsize'] = [12,6]
                graphe(temps,temperature,0.5)
```

```
        except:
            pass

    port_serie.close()
```

5.2.3 c. Animer une courbe :

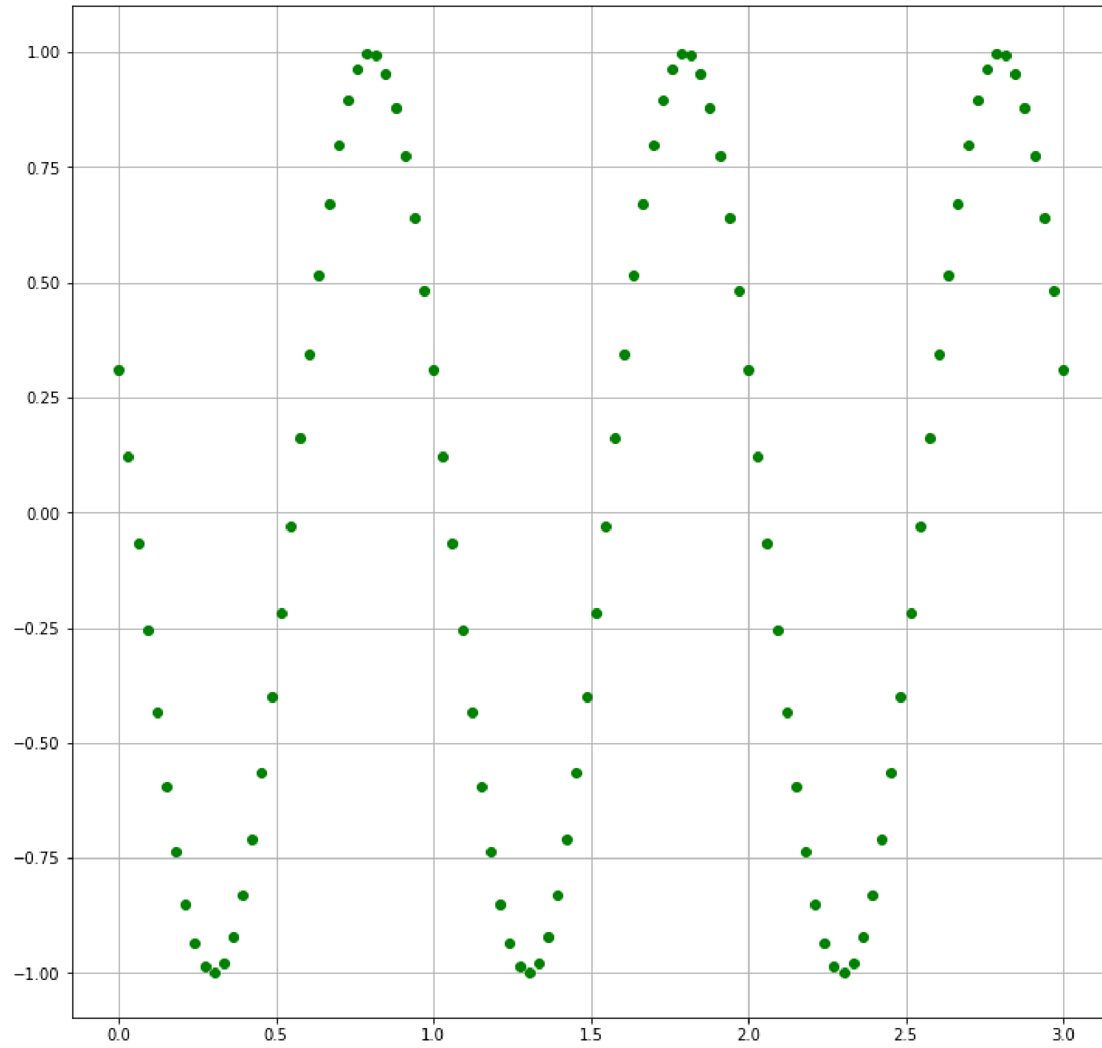
Le programme de première demande spécifiquement de *"simuler, à l'aide d'un langage de programmation, la propagation d'une onde périodique"*. Il faut pour cela animer la courbe.

Prenons l'exemple d'une onde périodique sinusoïdale d'amplitude $A = 1$ U.A., de fréquence $f = 1$ Hz et de longueur d'onde $\lambda = 1$ m.

L'amplitude est une fonction périodique du temps et de l'espace d'expression : $u(x,t) = A \cdot cos\left(2\pi \cdot \frac{x}{\lambda} - 2\pi \cdot \frac{t}{T}\right) = A \cdot \cos\left(k \cdot x - \omega \cdot t\right)$.

On obtiendra une simulation en traçant la fonction $u(x,t)$ pour des valeurs de t que l'ont fera varier dans une boucle ittérative.

```
In [50]: import numpy as np
         import matplotlib.pyplot as plt
         import time
         from IPython import display

         ### On définit d'abord les abscisses et les différents paramètres : ###

         x = np.linspace(0, 3, 100)
         l = 1.
         T=1.
         k = 2*np.pi/l
         omega = 2*np.pi/T
         dt = 0.1 # pas temporel

         t = 0
         Dt = 100

         for i in range(Dt):
             u = np.cos(k*x - omega*t)                       # On calcule u pour la valeur t.

             if i == 0:
                 plt.grid()
                 plt.rcParams['figure.figsize'] = [20,5]   # À la première boucle (t = 0),
                 line, = plt.plot(x, u,'go')                # on génère le graphe u(x).
             else:
                 line.set_ydata(u)                           # Pour les dates t suivantes, on met à
                 plt.draw()                                  # on rafraichit la figure
                 t = t + dt                                  # et on incrémente le temps t du pas d

             display.display(plt.gcf())                      # On utilise la combianaison de comman
             display.clear_output(wait=True)                 # vue au 3.a.
```




```
In [ ]:
```