# Python pour la SPC au lycée

*Version 1.0.1*

**GRP Python**

**févr. 05, 2020**

# Table des matières

CHAPITRE 1

# Introduction et installation

## 1.1 Introduction

### 1.1.1 Contexte

Dans le cadre de la réforme du lycée, les nouveaux programmes de physique-chimie mis en place à la rentrée 2019 introduisent un nouveau challenge pour la discipline, à savoir **proposer des activités qui impliquent les élèves dans la programmation et le codage**. Ces termes sont à comprendre en un sens très large : le codage peut consister à écrire complètement le programme (le langage Python est préconisé), mais peut également se limiter à l'adaptation d'un code existant, en modifiant les paramètres expérimentaux par exemple ou la précision attendue des résultats.

Les enseignants de physique-chimie sont déjà impliqués depuis près de trente ans dans les usages pédagogiques du numérique, d'une part dans le traitement et l'acquisition de données expérimentales, certains dans la modélisation ou la simulation, et tous, dans l'utilisation de supports multimédia ou de logiciels dédiés pour préparer et animer leurs cours. L'idée nouvelle est de **donner du sens à la modélisation** d'un phénomène ou d'une loi, et que le logiciel utilisé ne soit pas simplement une « boîte noire », dont l'élève ne connait ni le fonctionnement intrinsèque, ni les hypothèses, ni les limites.

### 1.1.2 Objectifs

L'introduction raisonnée et limitée de la programmation dans les programmes de physique-chimie a deux objectifs majeurs :

— Donner une **image actualisée de l'activité des scientifiques**, non seulement dans les études supérieures mais aussi dans de nombreux métiers.

— Apporter une plus-value dans les apprentissages, en développant les **compétences clés de la démarche scientifique** : raisonnement logique, capacité d'analyse par décomposition d'un problème complexe, distinction entre paramètres et variables, validation d'un modèle avec ou sans ajustement nécessaire, etc.

### 1.1.3 Mise en œuvre

Pour celui qui n'a jamais programmé, ou qui a de vagues souvenirs de ses études, un **accompagnement** est nécessaire, et c'est ce que propose ce guide. Il débute avec l'installation du logiciel Python, qui envisage plusieurs environnements possibles selon les contextes d'établissement ; il explique les bases de la programmation, passe en revue les différentes fonctions du logiciel Python, **avec des exemples systématiquement empruntés à la**

**physique-chimie**. Puis il propose des activités en relation directe avec les programmes de seconde et de première (voir capacités exigibles ci-dessous). Enfin il fournit un mémento, sur la syntaxe Python et sur le notebook Jupyter, pour que le professeur ait accès directement et rapidement à des éléments clés de la programmation, lorsqu'il prépare son cours.

Capacités exigibles

- Représenter les positions successives d'un système modélisé par un point, lors d'une évolution unidimensionnelle ou bidimensionnelle (2de)
- Représenter des vecteurs vitesse d'un système modélisé par un point, lors d'un mouvement (2de)
- Représenter un nuage de points associé à la caractéristique d'un dipôle, et modéliser la caractéristique de ce dipôle (2de)
- Déterminer la composition de l'état final d'un système siège d'une transformation chimique totale (1ère générale)
- 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 (1ère générale)
- Effectuer le bilan énergétique d'un système en mouvement (1ère générale)
- Représenter un signal périodique et illustrer l'influence de ses caractéristiques (période, amplitude) sur sa représentation (1ère générale)
- Simuler la propagation d'une onde périodique (1ère générale)

### 1.1.4 Réserves et perspectives

Les objectifs du programme se veulent restreints, l'accent est mis sur la **plus-value** que la programmation peut apporter en termes **d'apprentissage et d'assimilation de notions de physique-chimie**. Le but n'est pas de développer les compétences en codage des élèves, mais plutôt de s'appuyer sur les acquis de la formation qu'ils reçoivent par ailleurs, dans le cadre d'enseignements spécifiques ou au sein de certaines disciplines. Le recours à des tableurs ou à des logiciels de géométrie dynamique peut également conduire à des activités que l'on peut considérer comme de la programmation si l'élève conçoit lui-même de façon algorithmique, l'organisation des calculs ou la succession des opérations à réaliser.

Ce guide ne se veut pas exhaustif ni modélisant, c'est une aide pour démarrer avec Python. De nombreuses ressources existent sur le web, notamment des MOOC très bien conçus, pour aller plus loin. Chaque enseignant pourra, selon ses goûts et affinités, approfondir avec le temps, pour lui ou avec ses élèves, toutes les possibilités de mise en œuvre de la programmation dans son enseignement.

## 1.2 Environnement python

### 1.2.1 Introduction

Le `Python` est un langage **interprété**, c'est à dire que les instructions écrites dans un simple fichier texte par l'utilisateur sont ensuite "traduites" par le programme `Python` en instructions exécutables par l'ordinateur.

Il est donc nécessaire de disposer de cet interpréteur pour pouvoir exécuter un programme écrit en python.

Même si un simple fichier texte permet d'écrire un programme python, il existe un environnement beaucoup plus sympathique, avec de grandes possibilités pédagogiques : les **Notebooks Jupyter**. Nous avons fait le choix de cet outil pour les sciences physiques.

Si votre établissement dispose de l'ENT Pentila Nero (celui auquel on accède via www.l-educdenormandie.fr), vous avez déjà un environnement Python/Jupyter prêt à être utilisé, avec les fonctionnalités utiles de l'ENT (groupes, partager un document, accès maison / établissement, casier pour déposer / rendre un travail...)

Si ce n'est pas le cas, condoléances, vous allez devoir passer par l'étape "installation de python", un peu plus loin dans ce document.

### 1.2.2 Python via l'ENT

Pour utiliser les fonctionnalités python via votre ENT, connectez vous sur votre ENT "comme d'habitude".

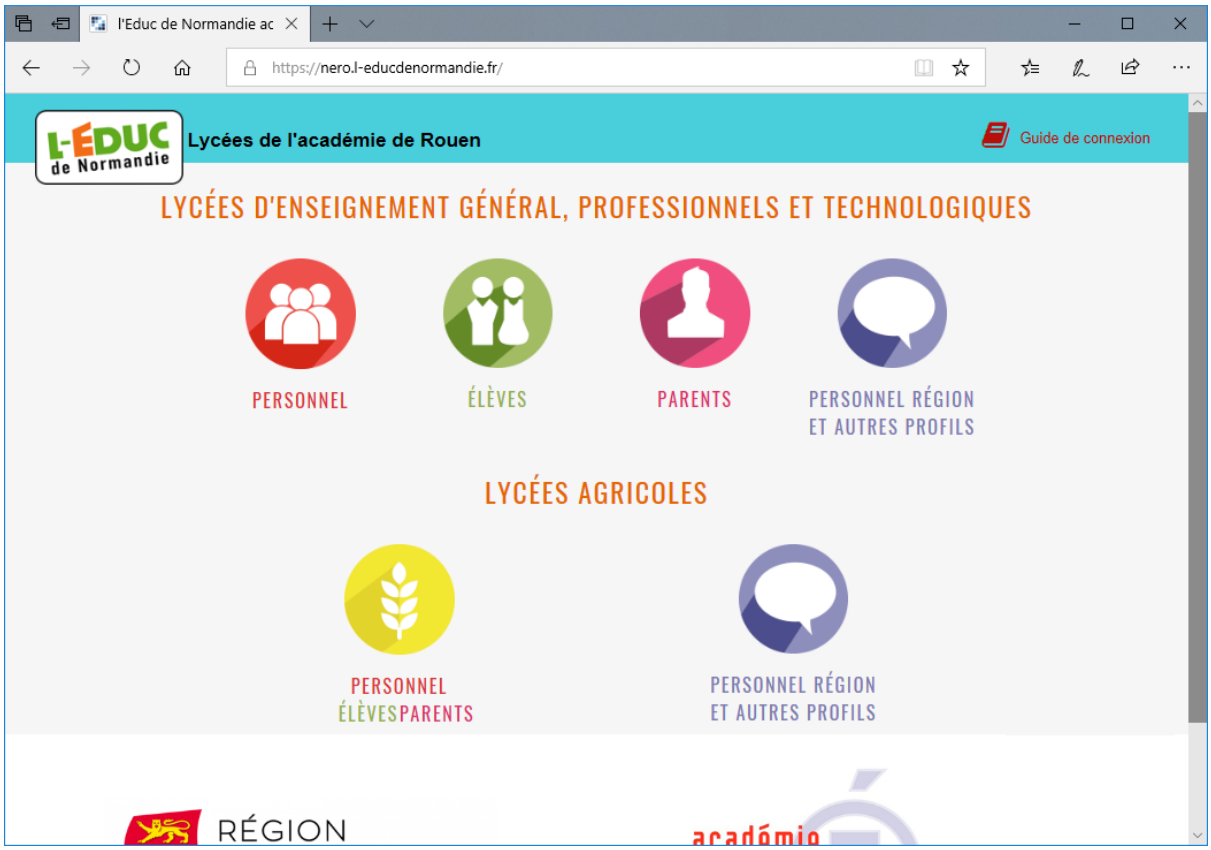



Fig. 1 – Connexion Nero

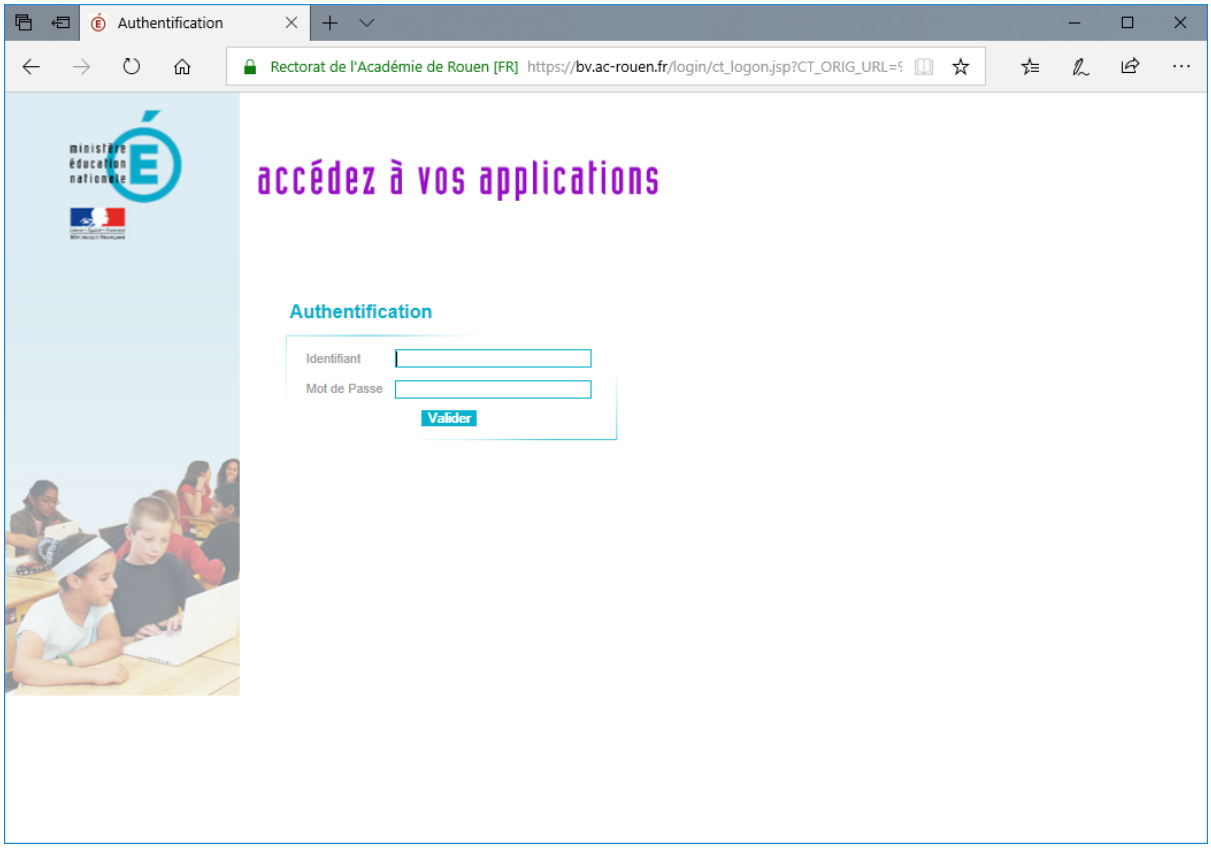



Fig. 2 – Connexion académique

Naviguez ensuite vers votre espace “Mes documents”

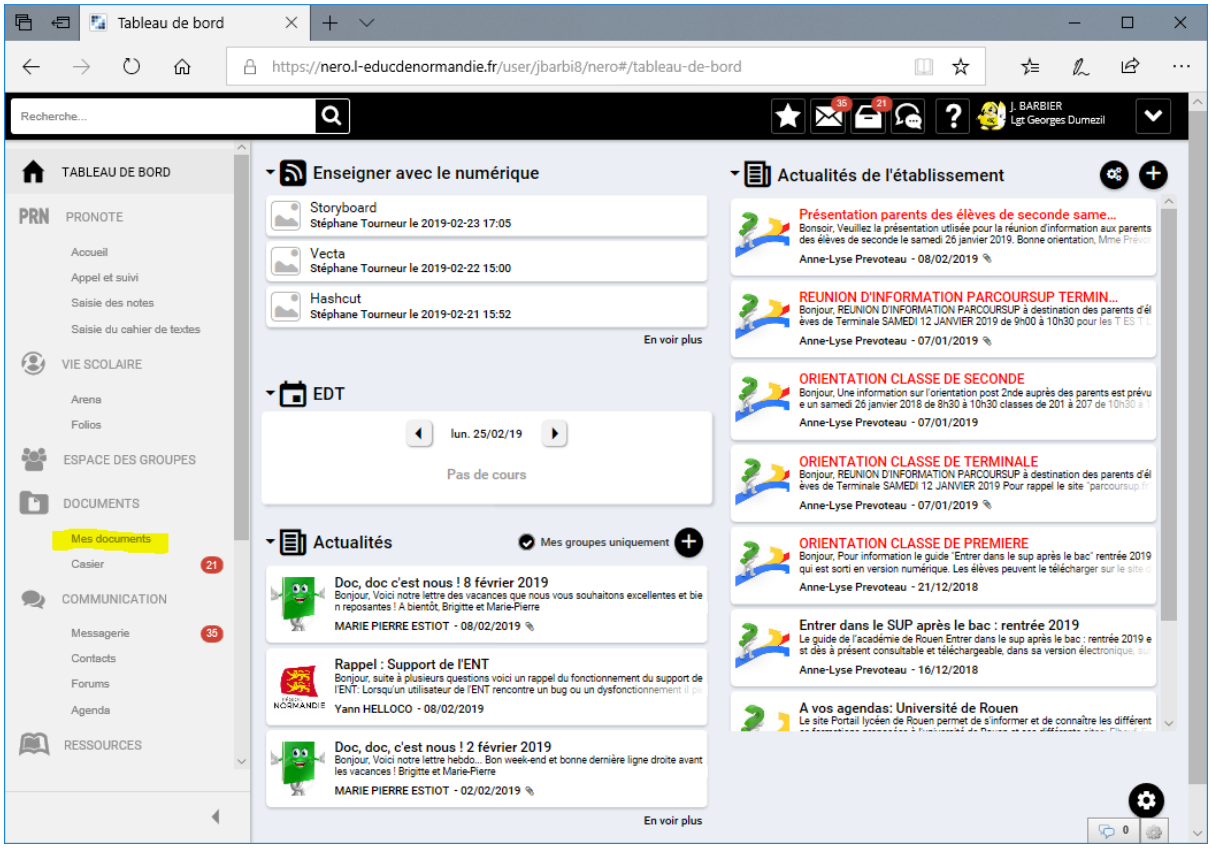



Fig. 3 – Mes documents

Cliquez ensuite sur le bouton “Créer” (ou bien via un clic droit de la souris) : dans la liste des formats de fichier disponible, vous devriez voir apparaître “Jupyter Notebook”.

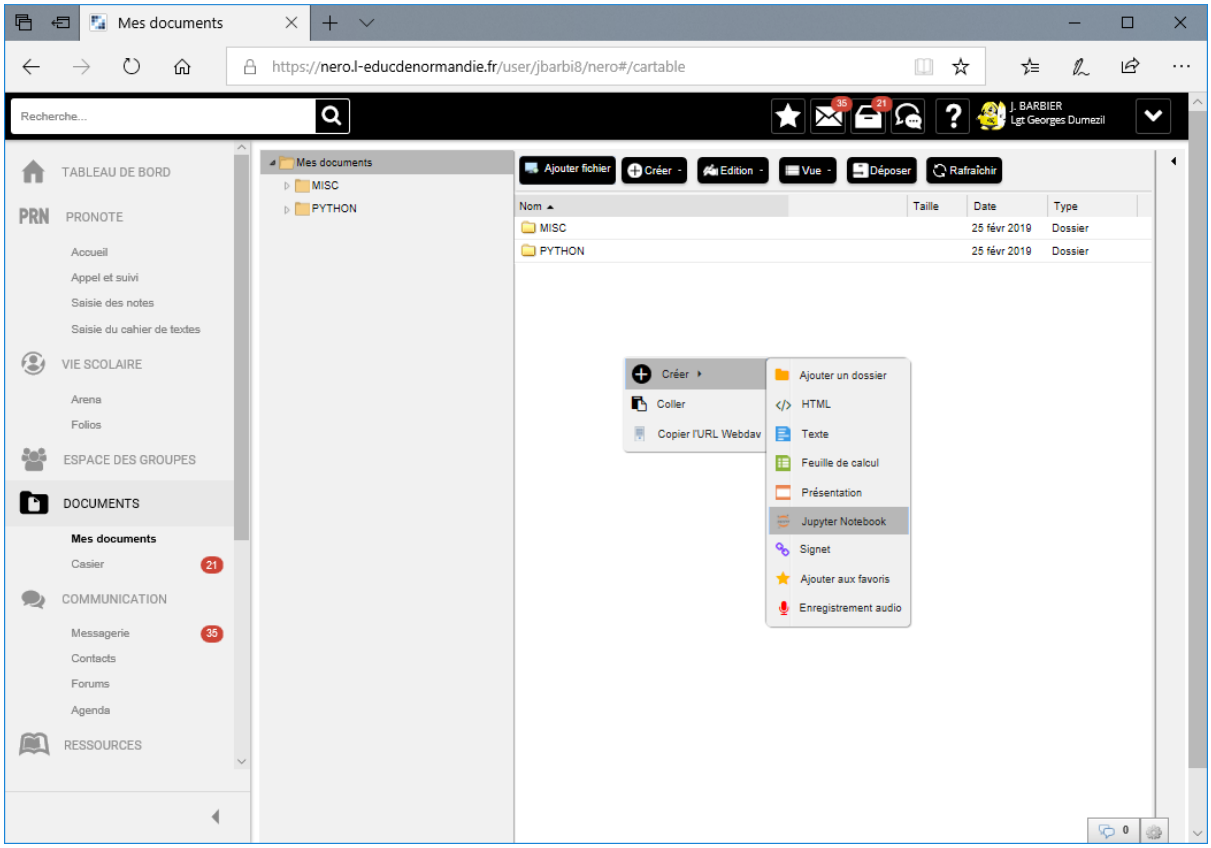



Fig. 4 – Créer un notebook 1

Si “Jupyter Notebook” n’apparaît pas, il faut demander à l’administrateur ENT de votre établissement de l’activer. La procédure prend moins d’une minute, les détails sont à la fin de ce document).

Si il apparaît, cliquez dessus, et nommez votre fichier. Une fenêtre apparaît alors, c’est votre nouveau notebook.

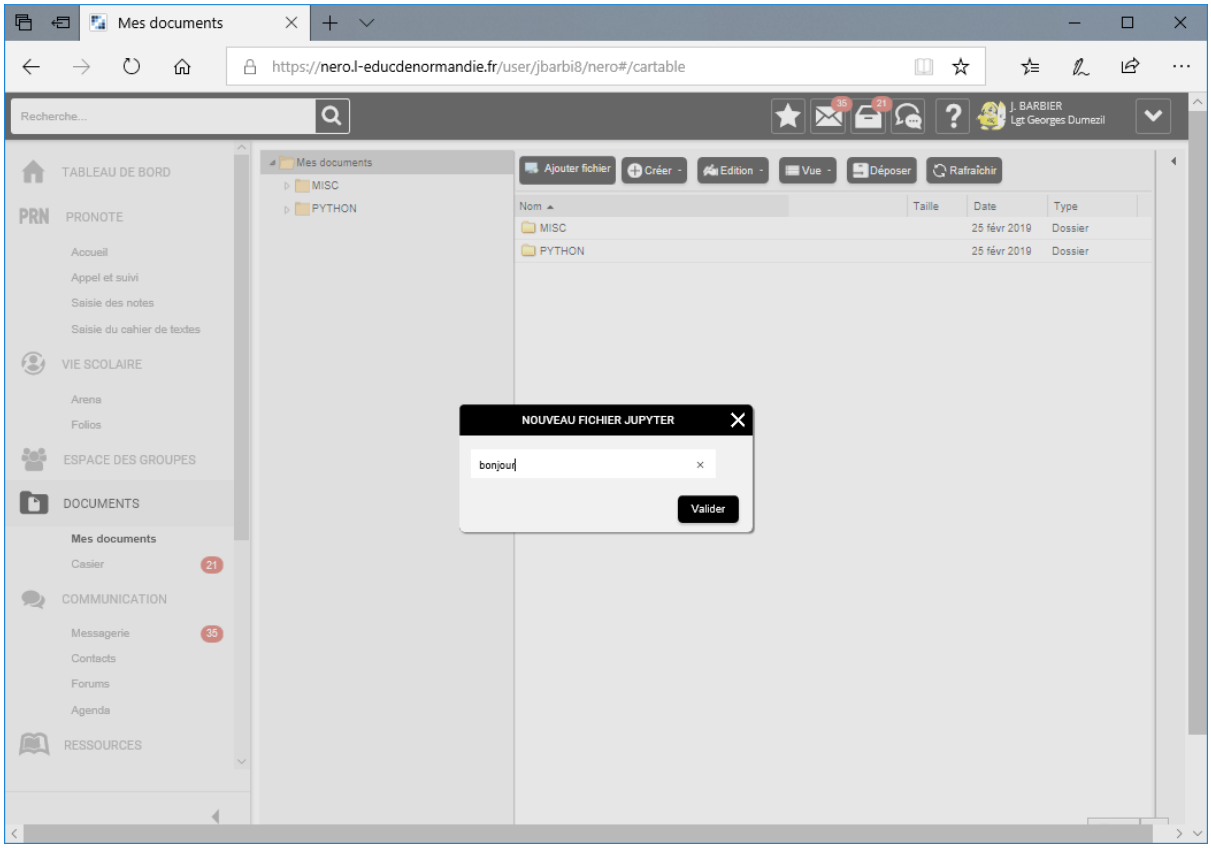



Fig. 5 – Créer un notebook 2

Vous alors pouvez taper

```
print("Bonjour le monde")
```

dans la cellule “In”.

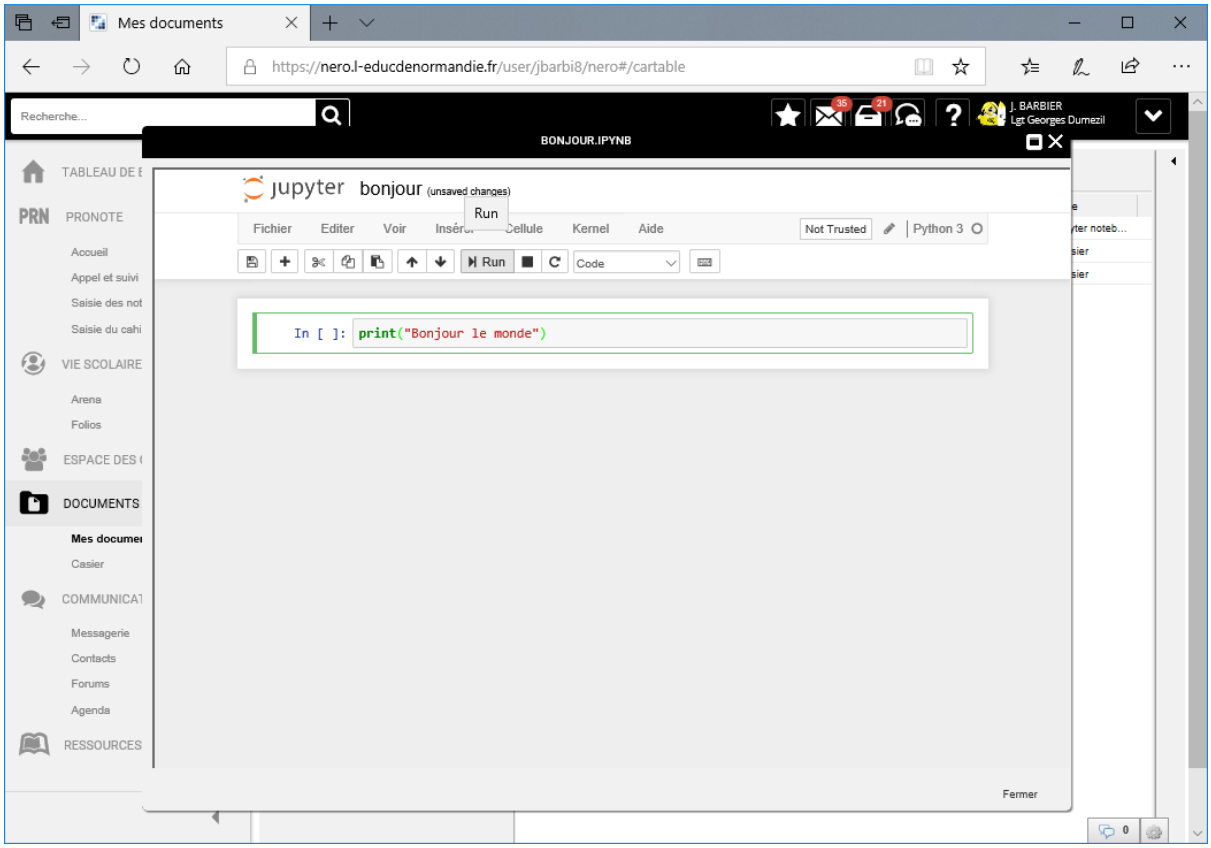



Fig. 6 – Créer un notebook 3

Exécutez ensuite ce premier programme en cliquant sur le bouton “Run”. Vous devriez voir apparaître la phrase “Bonjour le monde” juste en dessous.

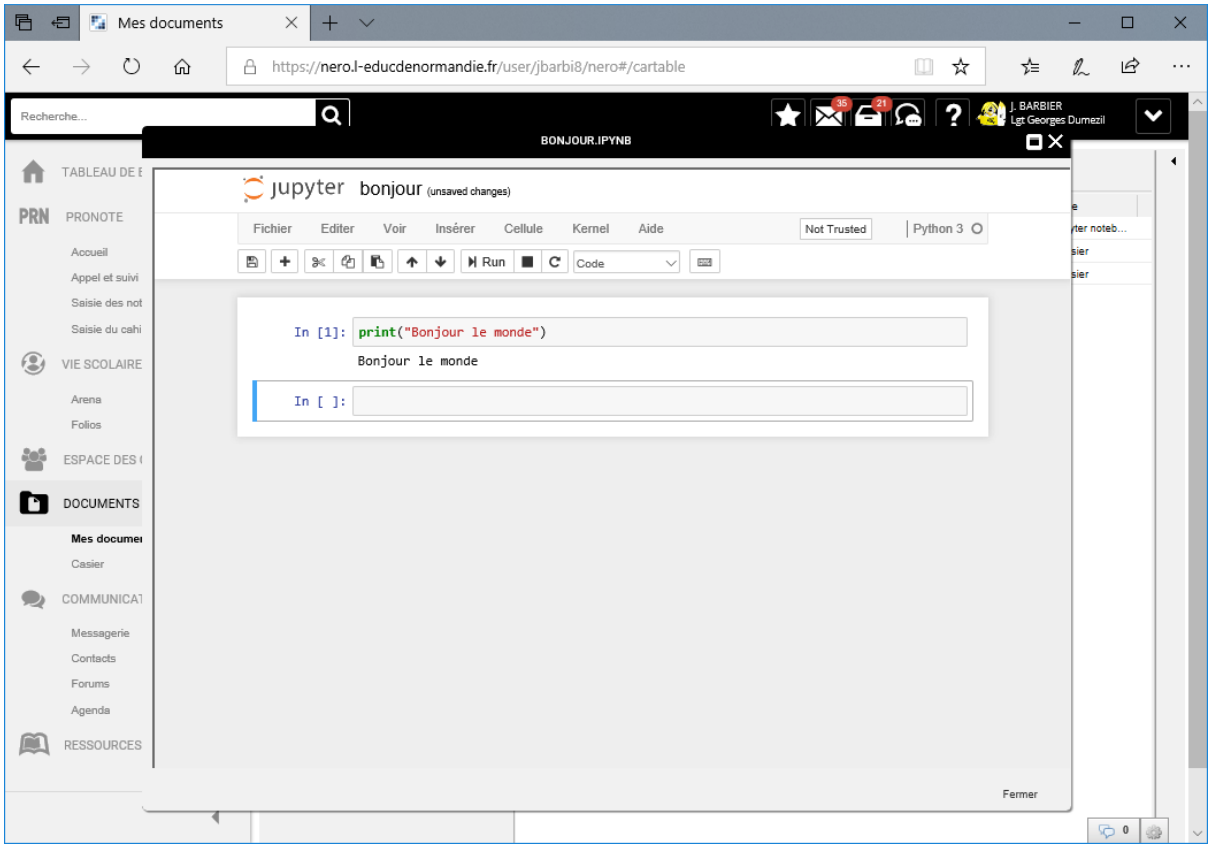



Fig. 7 – Créer un notebook 4

Vous avez exécuté votre premier programme python. Vous pouvez maintenant passer à la suite de ce document, en sautant éventuellement les deux chapitres suivant.

Remarque : depuis la rentrée 2019, une autre interface existe pour les notebooks jupyter : **jupyter lab**. Si l'administrateur de votre ENT l'a activée (c'est une simple case à cocher), vous pouvez lancer cette interface par l'entrée de menu **JupyterLab** en dessous de **Casier**

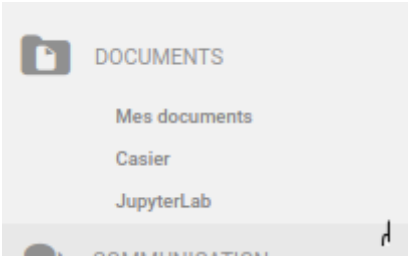



Fig. 8 – Lancer jupyterlab

Cette interface facilite le travail avec plusieurs documents et possède quelques fonctionnalités en plus. Mais l'essentiel n'est pas fondamentalement différent.

### 1.2.3 Installation de python sous Windows

Le programme `Python` seul, que l'on trouve sur le site python.org n'est donc pas très gros, quelques méga-octets au plus. Il en existe plusieurs générations, que l'on repère à leur numéro de version : `2.x` ou `3.x`. Il est fortement recommandé d'utiliser la génération `3.x` (3.7 aujourd'hui), même si beaucoup d'exemples que l'on peut trouver sur internet sont de la génération `2.x`.

`Python` seul a déjà beaucoup de fonctionnalités à la base. Mais ce qui fait sa richesse, c'est le nombre énorme de "paquets" qu'on peut lui ajouter (des fonctionnalités créées par d'autres personnes et qu'on peut directement réutiliser).

Il n'est pas évident de rajouter à la main un paquet sous windows, c'est pour cela que plusieurs *distributions* de python (le programme python + un choix de paquets) sont disponibles. Chaque distribution a ses avantages et ses inconvénients. Nous avons fait le choix d'utiliser une des plus complètes, et qui a l'avantage de gérer directement les notebooks Jupyter : anaconda.

Cette installation peut être faite sur un ordinateur personnel (à la maison), sur des ordinateurs d'une classe mobile ou d'une salle info. La solution ENT est à privilégier, mais l'installation locale peut permettre de s'adapter à des situations particulières (pas de réseau / problèmes de réseau / pas d'ENT / . . . )

#### 1.2.3.1 Installation

Les instructions suivantes sont pour une installation sous Windows 10, 64 bits, sur un système avec au moins 3 Go de disque disponible.

Si une autre version de python est déjà présente sur le système, il vaut mieux la désinstaller au préalable. Anaconda n'est pas intrusif, mais d'autres distributions le sont plus et peuvent poser un problème de cohabitation.

**Etape 1** Télécharger anaconda

Sur le site anaconda.com, bouton **Download** en haut à gauche. Choisir la version correspondant à votre système (32/64 bits).

Le téléchargement prend un certain temps (+ de 600 Mo).

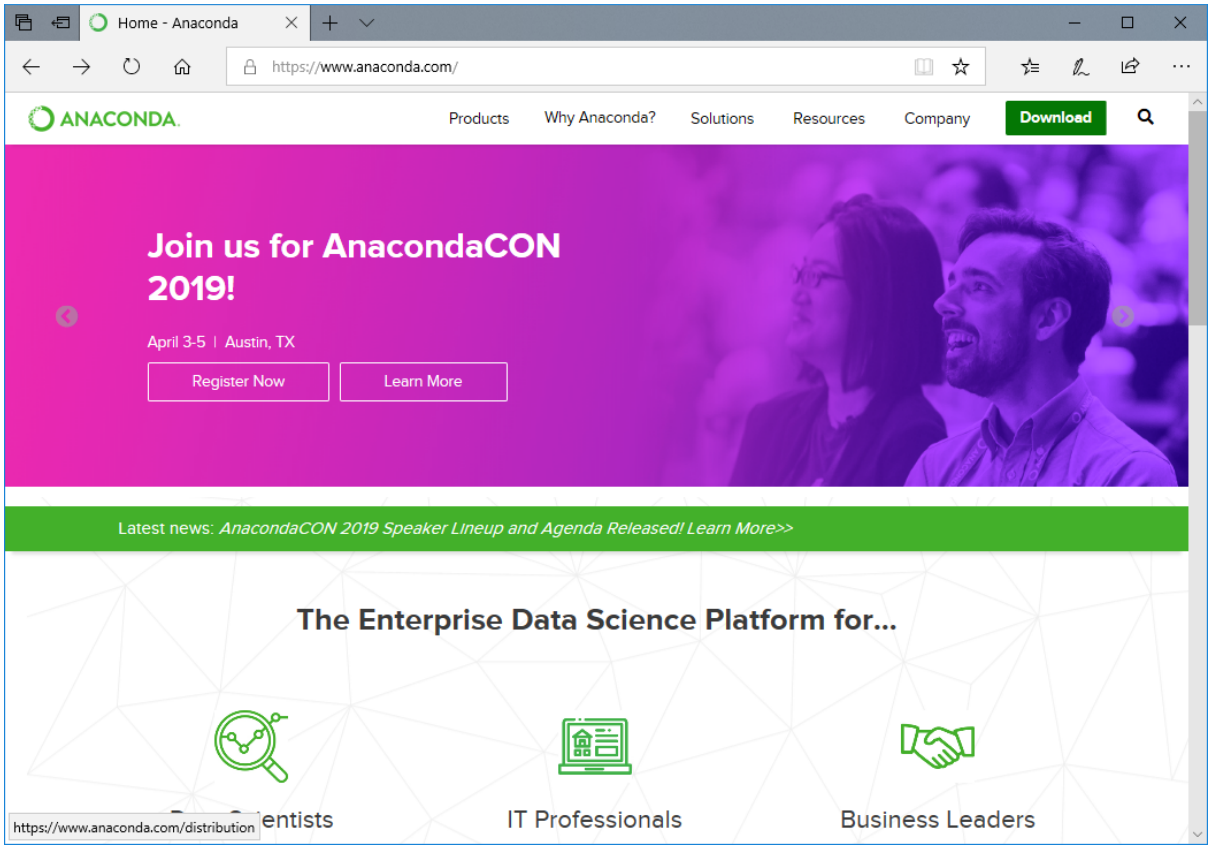



Fig. 9 – Installer anaconda 01

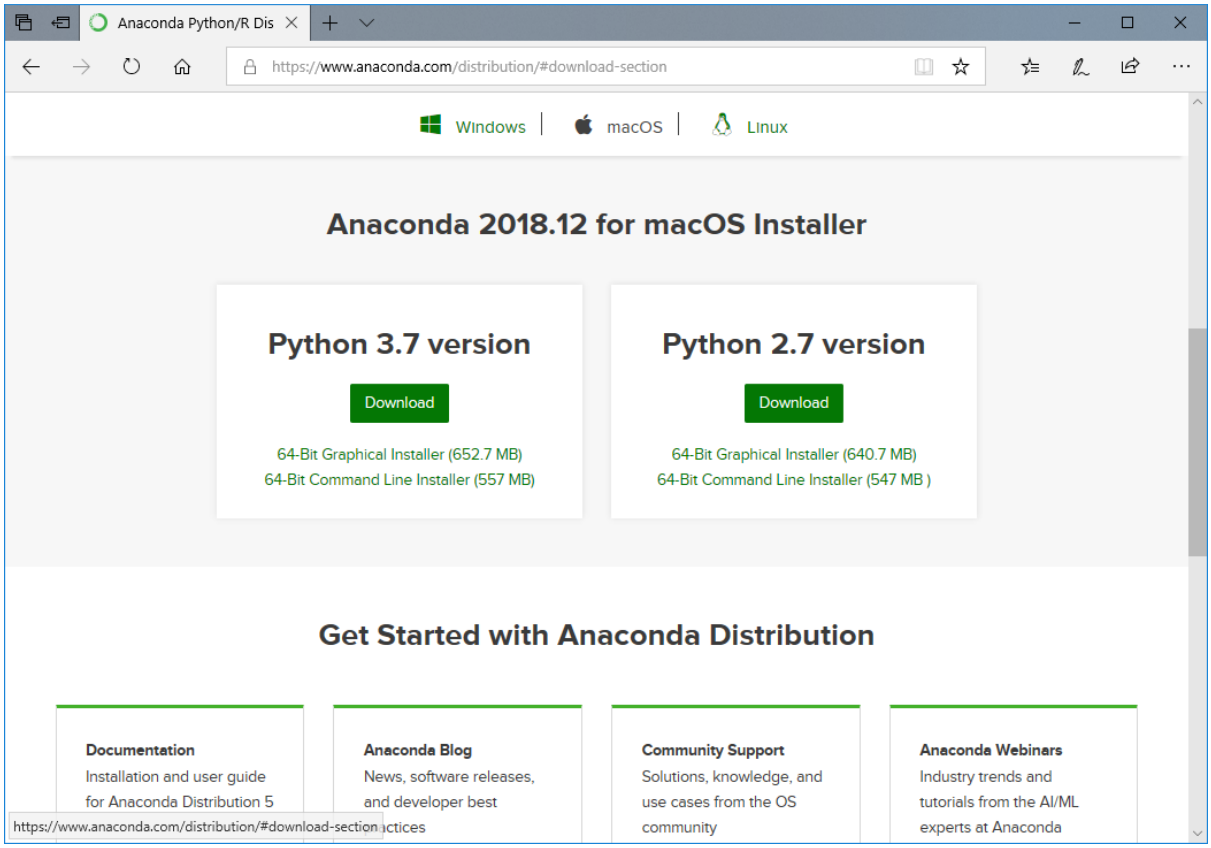



Fig. 10 – Installer anaconda 02

**Etape 2** Installer anaconda

Lancer le programme d'installation ; les valeurs par défaut des différentes étapes sont suffisantes, en gros :

- accepter la licence
- choisir une installation "mono-utilisateur" (juste pour vous)
- choisir le dossier d'installation (si possible, un chemin sans espaces est préférable)
- définir anaconda comme interpréteur python par défaut

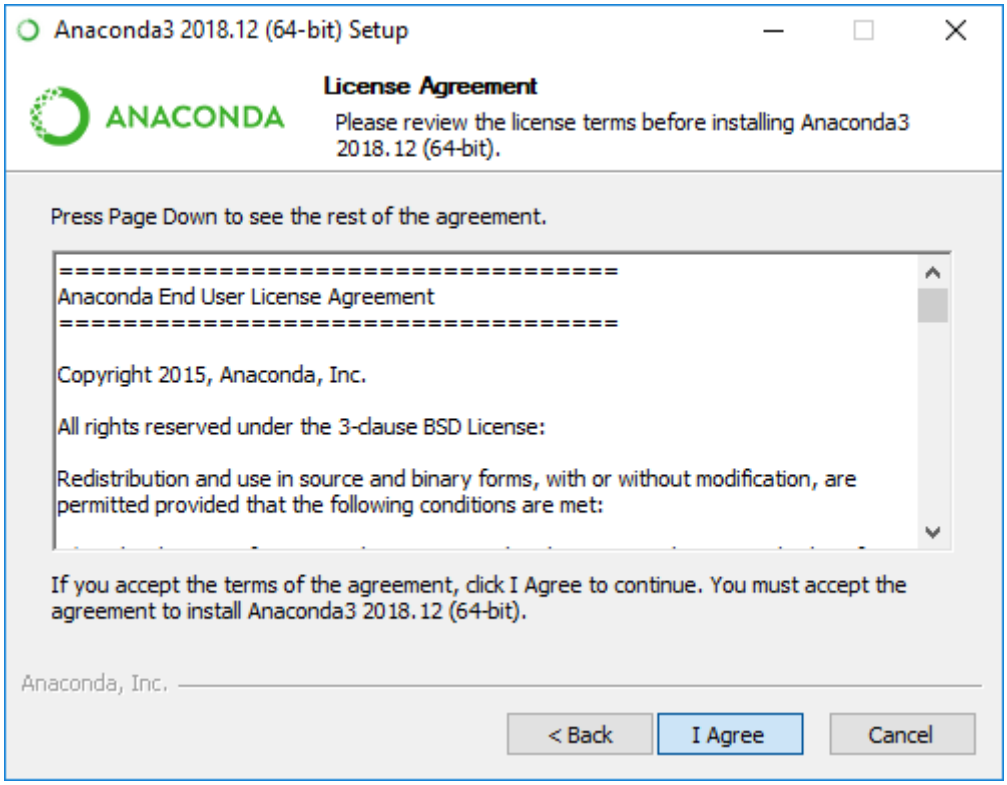



Fig. 11 – Installer anacondat 03

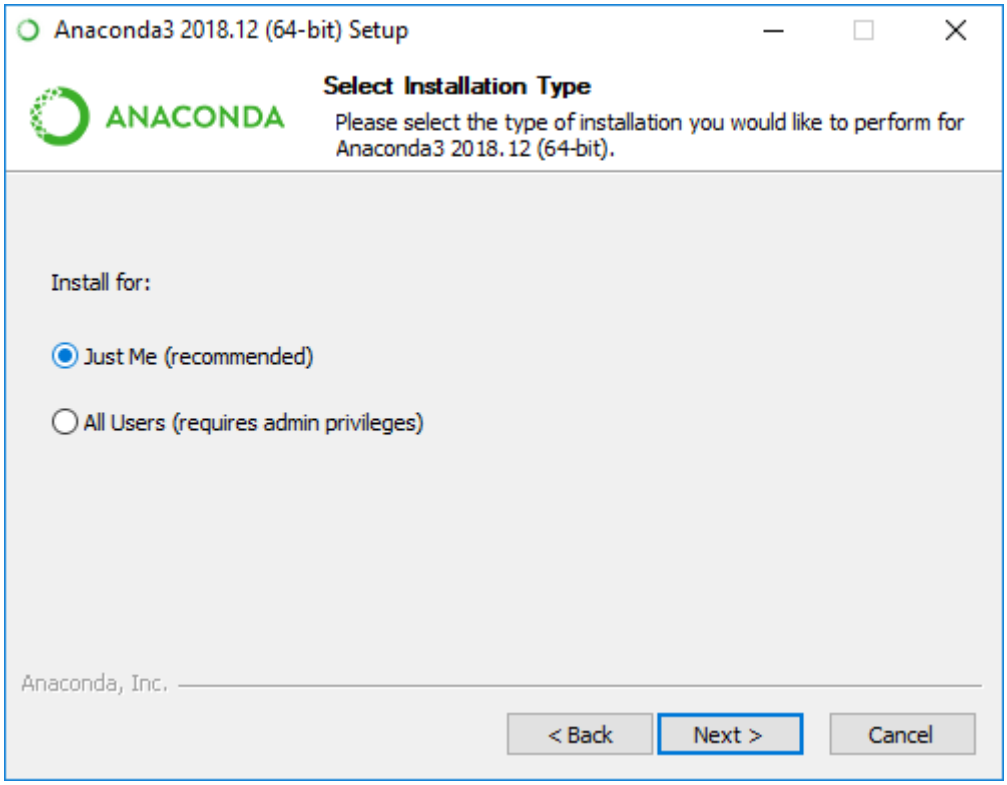



Fig. 12 – Installer anacondat 04

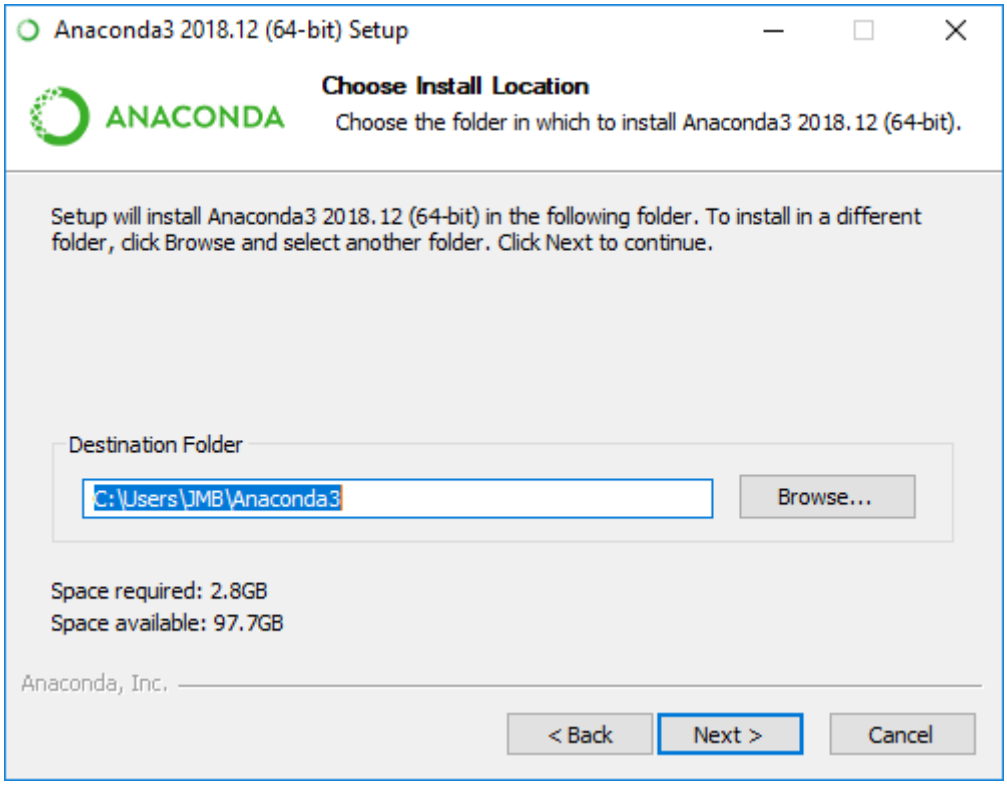



Fig. 13 – Installer anacondat 05

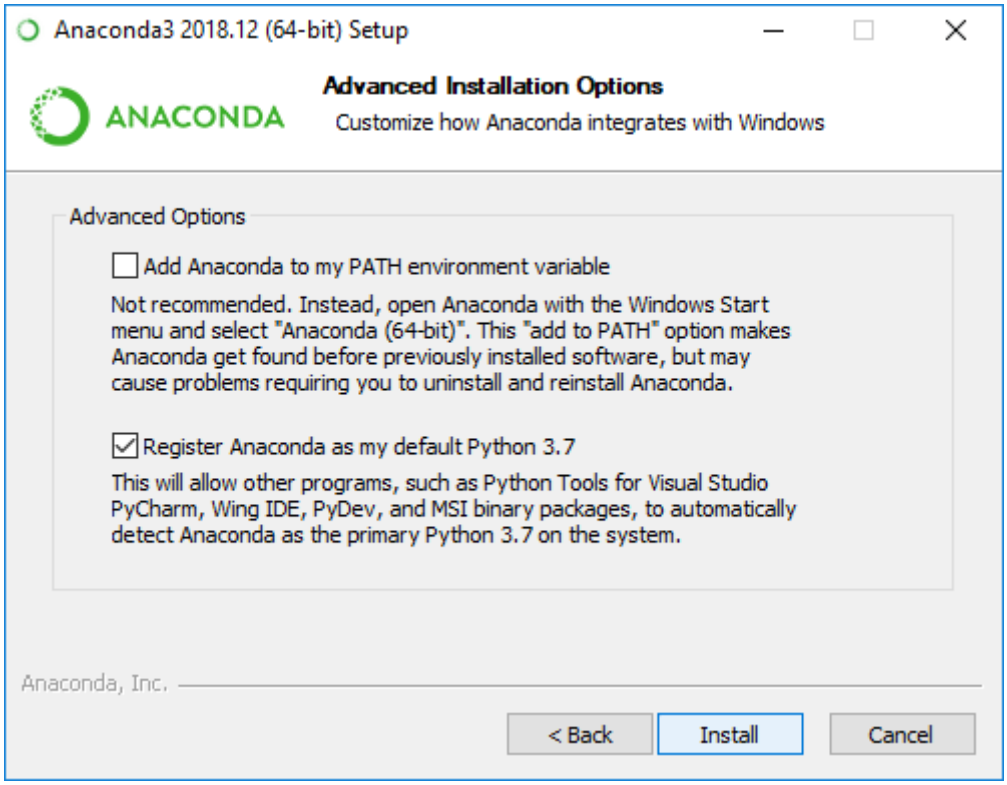



Fig. 14 – Installer anacondat 06

Puis attendre la fin de la copie des fichiers (qui prend aussi un certain temps)

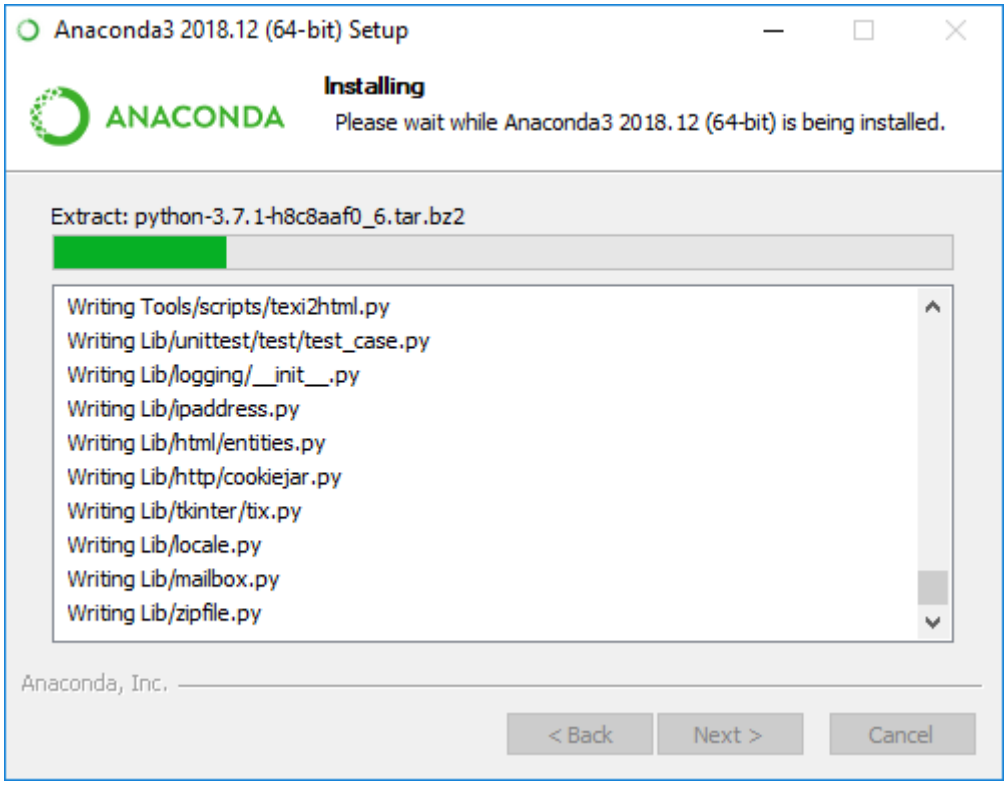



Fig. 15 – Installer anaconda 07

Une fois la copie des fichiers terminées et l'apparition de quelques fenêtres noires (qu'il ne faut pas fermer, elles se ferment toute seules quand elles ont terminé leur travail), l'installateur propose l'installation de Visual Studio Code (un éditeur de code d'assez bonne qualité). Vous pouvez l'installer ou non, au choix, nous utiliserons un autre éditeur pour l'instant.

Puis il propose deux autres cases à cocher pour en apprendre plus, vous pouvez les décocher ou jeter un coup d'oeil aux ressources proposées.

Anaconda est alors installé.

#### 1.2.3.2 Exécution

Les différentes fonctionnalités d'anaconda sont regroupées dans une interface appelée *Anaconda Navigator*. Vous pouvez y accéder par le menu Démarrer, dans la rubrique *Anaconda*.

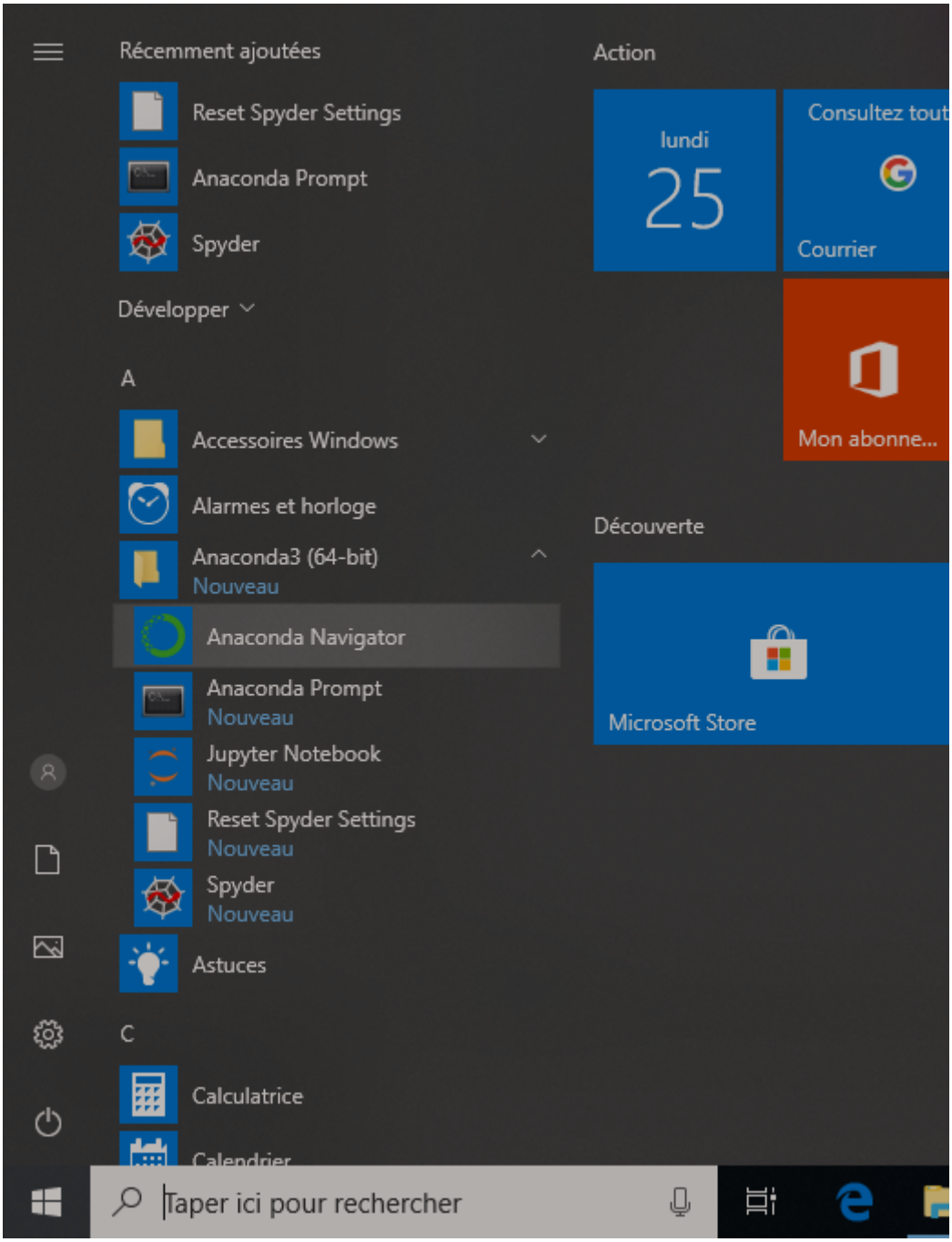



Fig. 16 – Exécuter anaconda

La page d'accueil d'Anaconda Navigator présente les différentes applications installées (Launch) ou installables (Install).

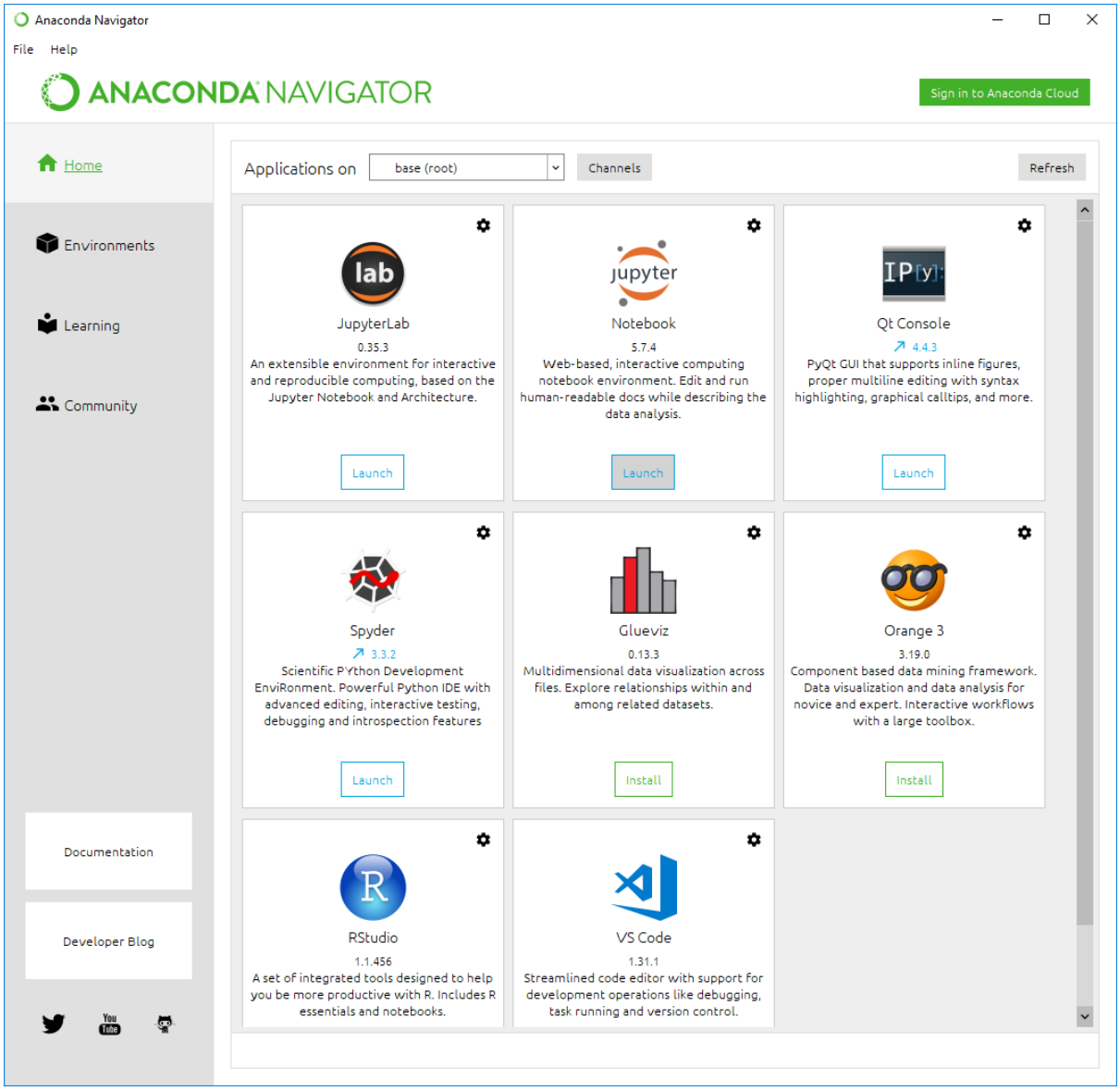



Fig. 17 – Exécuter anaconda

La plupart des exemples de ce guide sont des **Notebooks Jupyter**. Démarrer Jupyter en cliquant sur “Launch” ; au premier lancement, vous aurez éventuellement à choisir le navigateur de votre choix pour utiliser les notebooks. Choisissez ce que vous voulez sauf Internet Explorer.

Dans la fenêtre de navigateur qui s’ouvre, vous trouverez la liste de vos dossiers. Vous pouvez naviguer dans l’arborescence de vos documents ; choisissez le répertoire de votre choix, et une fois que vous êtes dedans, vous pouvez créer votre premier notebook : bouton “New” en haut à droite, choisir “Python 3”.

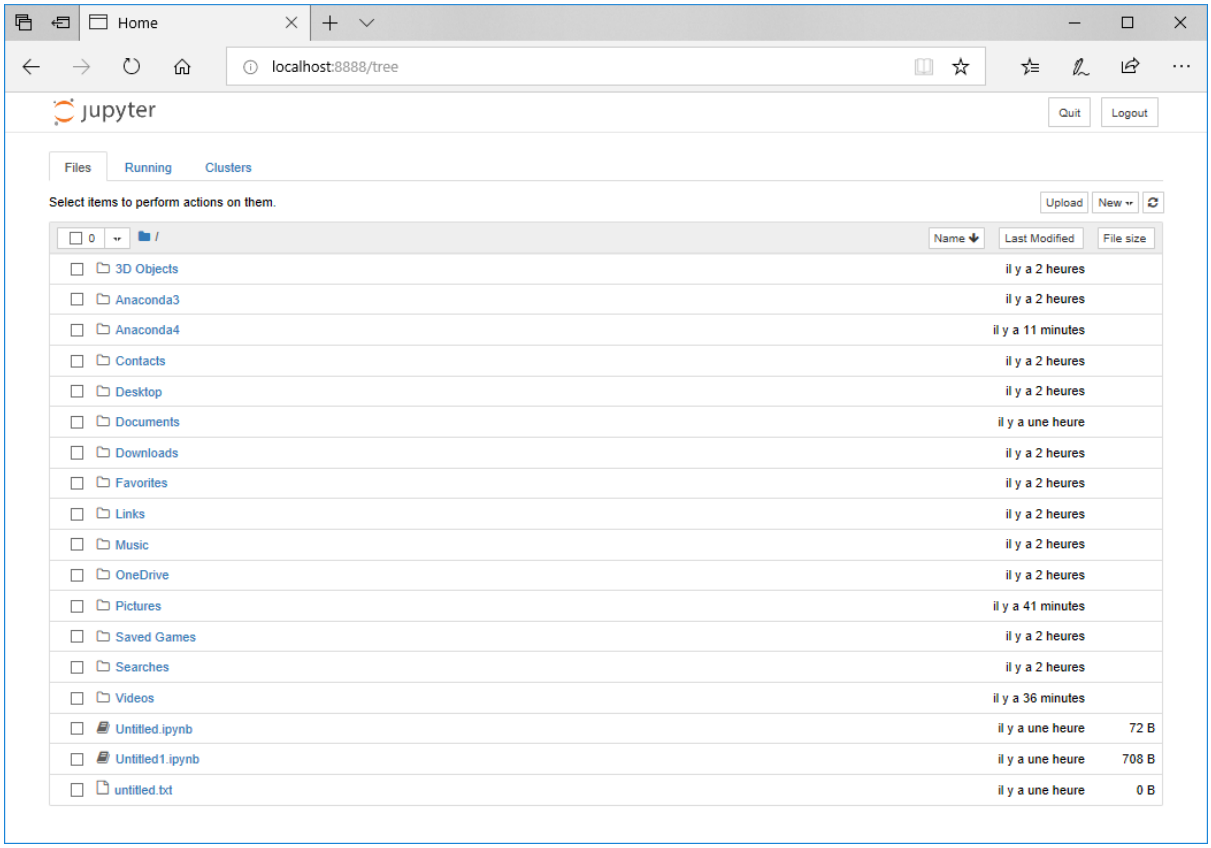



Fig. 18 – Exécuter anaconda

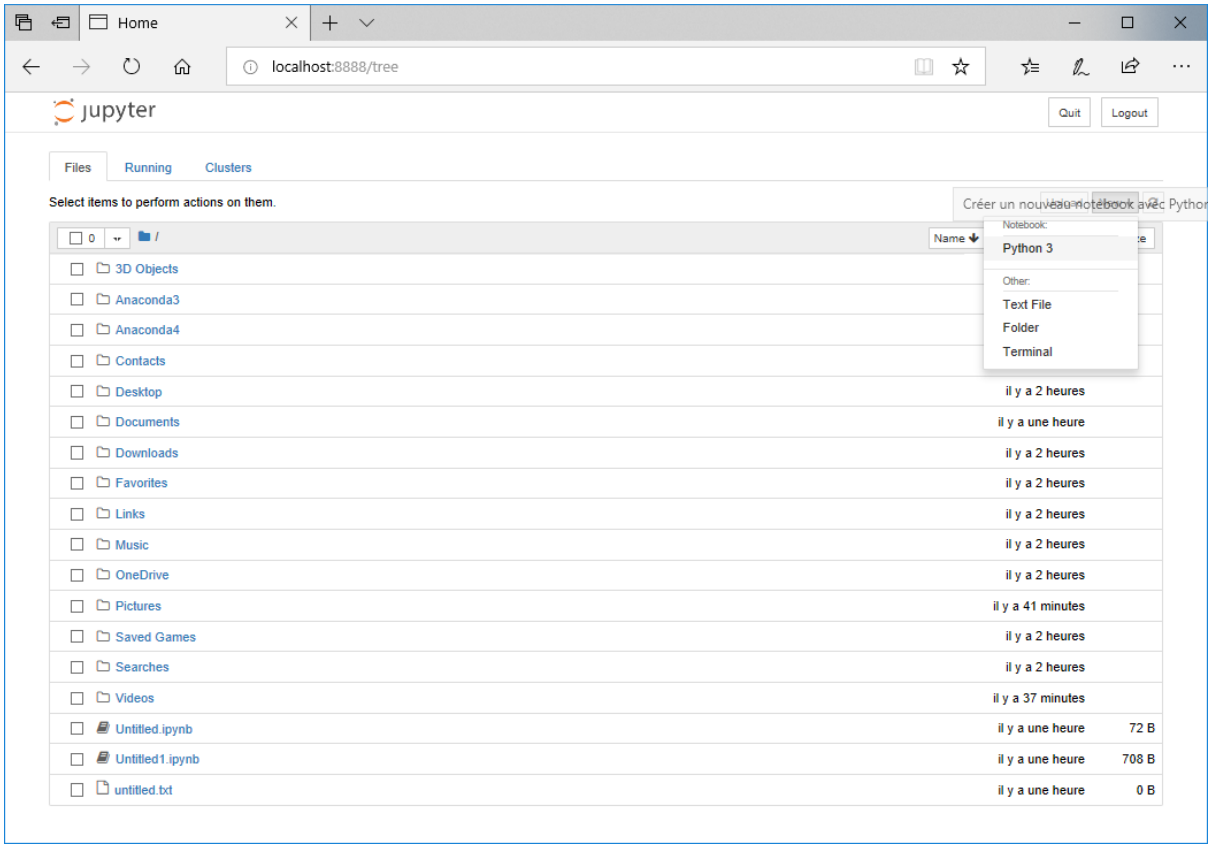



Fig. 19 – Exécuter anaconda

Un nouvel onglet s'ouvre alors, avec votre notebook.

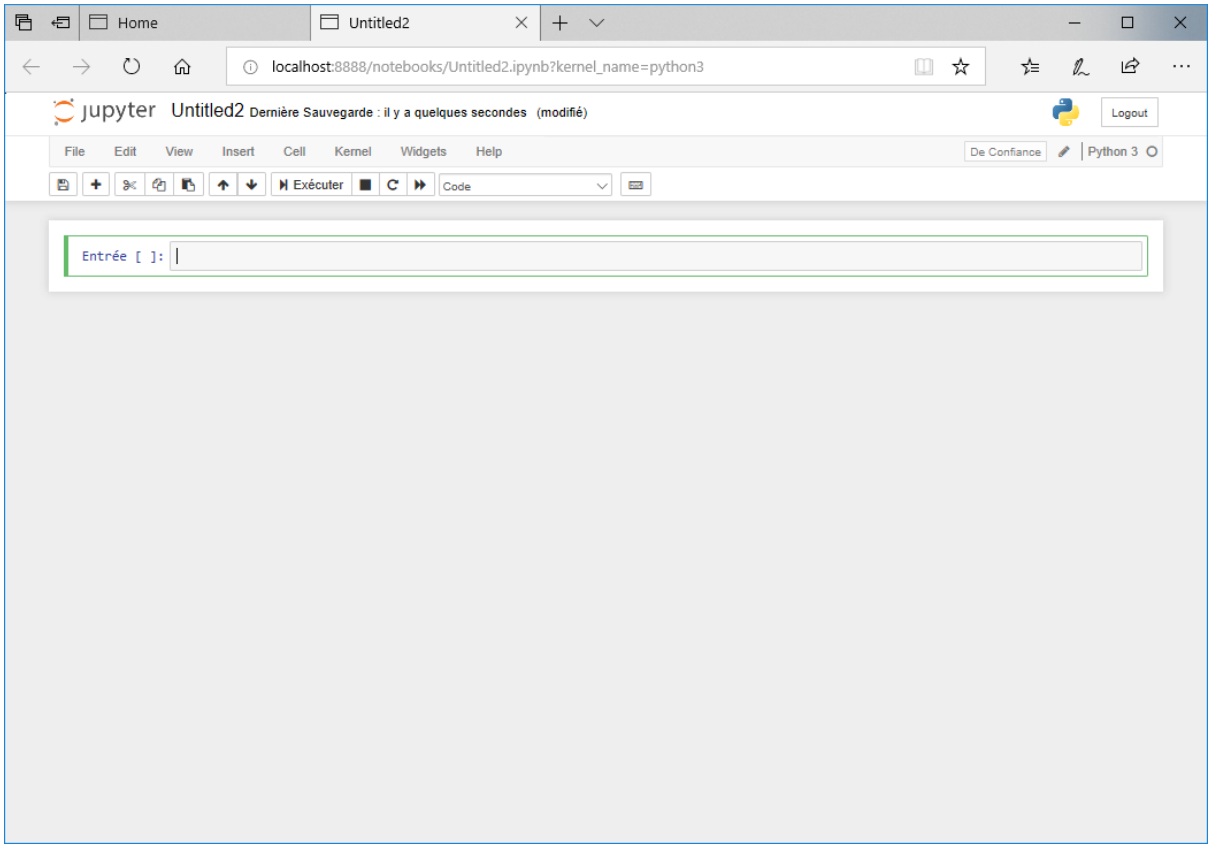



Fig. 20 – Exécuter anaconda

Pour modifier le titre du notebook, cliquez dessus une fois qu'il est ouvert. Dans la cellule "Entrée", vous pouvez écrire

```
print("Bonjour le monde")
```

et cliquer sur le bouton "Exécuter". Vous devriez voir apparaître la phrase "Bonjour le monde" juste en dessous.

Vous avez fait votre premier programme en python. Vous pouvez maintenant continuer à explorer les ressources de ce guide en utilisant cette installation.

### 1.2.4 Activation des notebooks sur Nero

Pour activer les notebooks sur Nero, en tant qu'administrateur de l'ENT :

1. Sélectionner dans la barre de navigation latérale « Administration > Applications »
2. Cliquer sur l'icône « Jupyter »
3. Sélectionner d'icône de paramètrage (la roue dentée)
4. Ajouter une règle de diffusion pour le public concerné

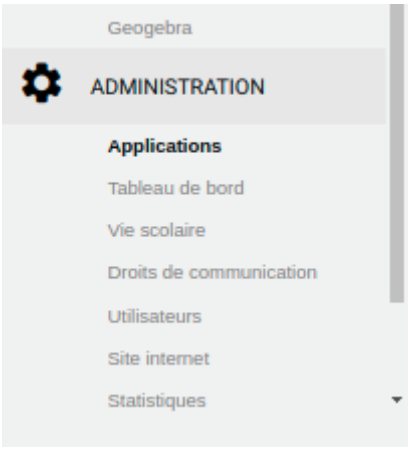



Fig. 21 – Activer les notebooks sur Nero : étape 1


Fig. 22 – Activer les notebooks sur Nero : étape 2

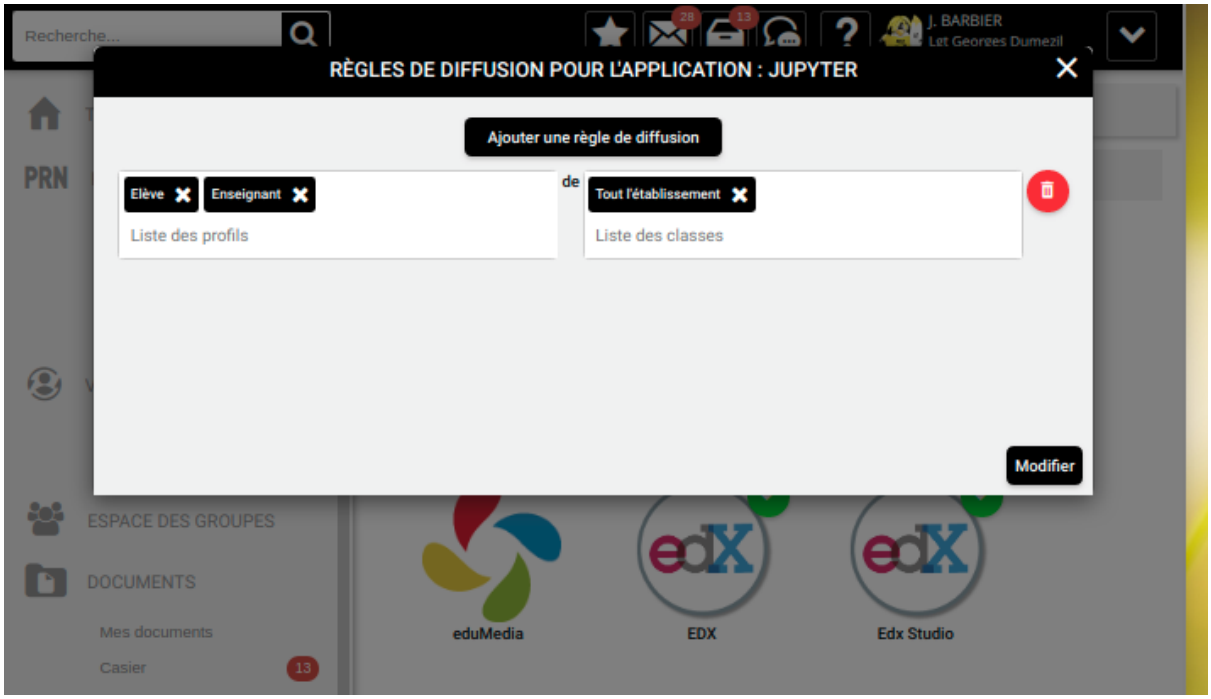



Fig. 23 – Activer les notebooks sur Nero : étape 3

## 1.3 Guide d'utilisation rapide du Notebook Jupyter

`Télécharger ce guide en PDF`

Après avoir téléchargé & installé Anaconda sur votre ordinateur :

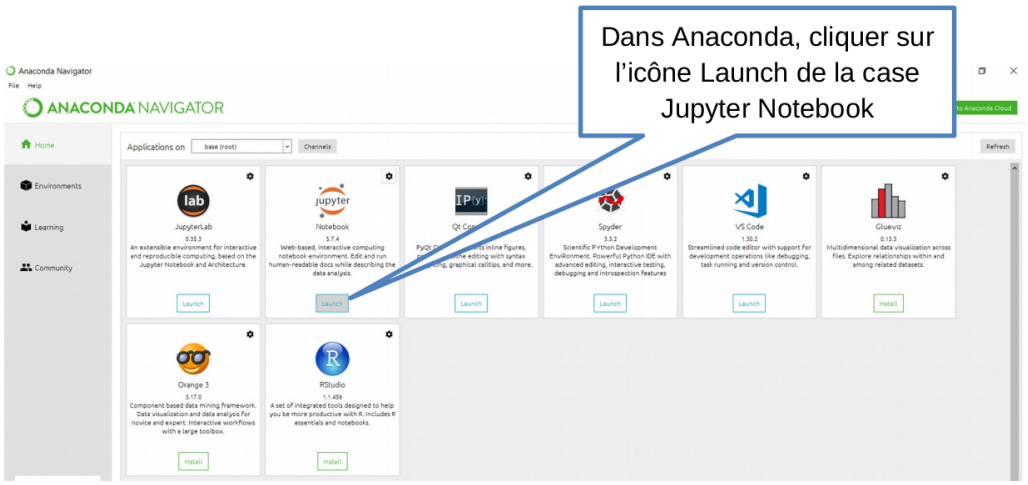



Fig. 24 – Lancement jupyter notebook

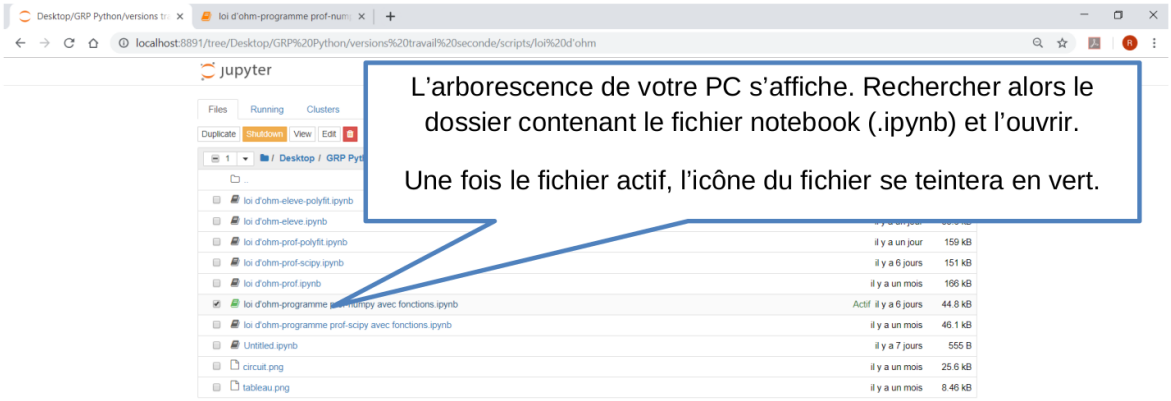



Fig. 25 – Arborescence jupyter notebooks

Remarque : Dans le cas où votre fichier Notebook se situe dans le dossier Mes documents de votre ENT, ouvrez directement le fichier en cliquant sur son nom.


Fig. 26 – Ouverture notebook

Un fichier notebook se présente comme une succession de cellules qui sont de deux types :

— des cellules « Markdown » permettant d'écrire du texte (titre, contexte de l'activité, consignes élèves...), d'insérer des fichiers image...

— des cellules « Code » permettant d'écrire des lignes de code en langage Python puis de les exécuter. Le mot « Entrée [ ] » (ou « In [] ») est écrit dans la marge à gauche dans ce type de cellule.

Copie d'écran avec des cellules « Markdown » et une cellule « Code ».

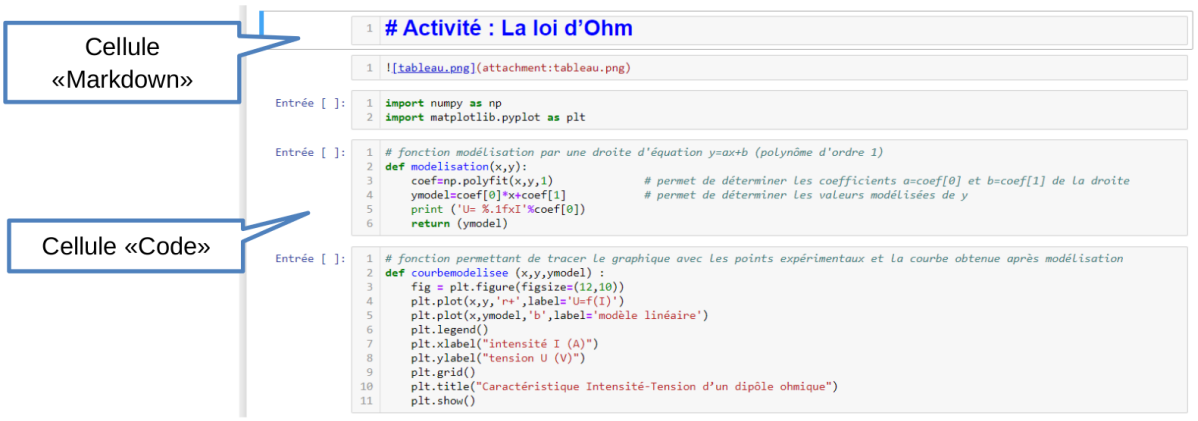



Fig. 27 – Types de cellules

### 1.3.1 Comment utiliser un fichier Notebook

Pour utiliser votre fichier Notebook, il faut exécuter les cellules du Notebook dans l'ordre, les unes après les autres.

Pour cela :

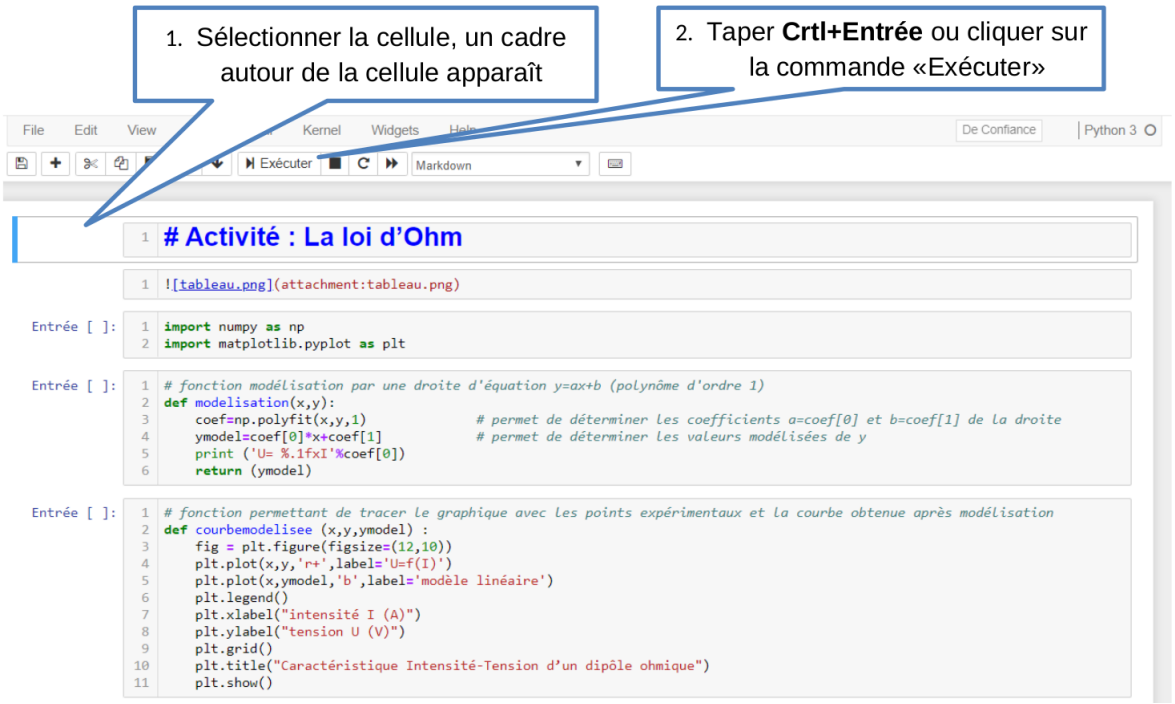



Fig. 28 – Exécuter un notebook

Copie d'écran avec les cellules « Markdown » et « Code » précédentes exécutées.

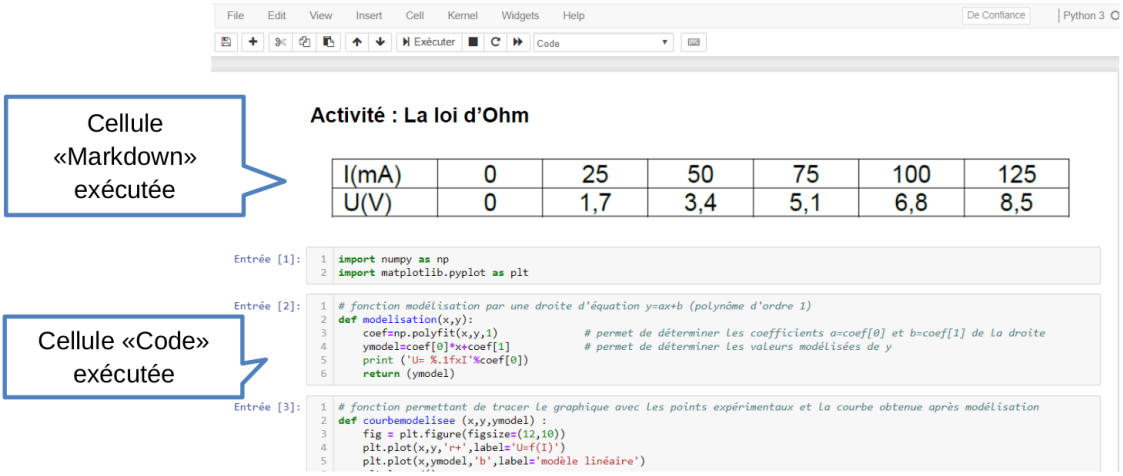



Fig. 29 – Cellules exécutées

— Si la cellule est de type « Markdown » : les textes, les images jointes… entrés dans la cellule apparaissent dans leur forme lisible par l'utilisateur du fichier Notebook. Lors de l'ouverture du fichier, ces cellules ont souvent déjà été exécutées par le créateur du fichier, ce qui permet de faire de belles présentations notamment à destination des élèves.
— Si la cellule est de type « Code » : Lors de l'exécution, une étoile * apparaît entre les crochets puis une fois la cellule exécutée, le « numéro d'ordre d'exécution » s'affiche.

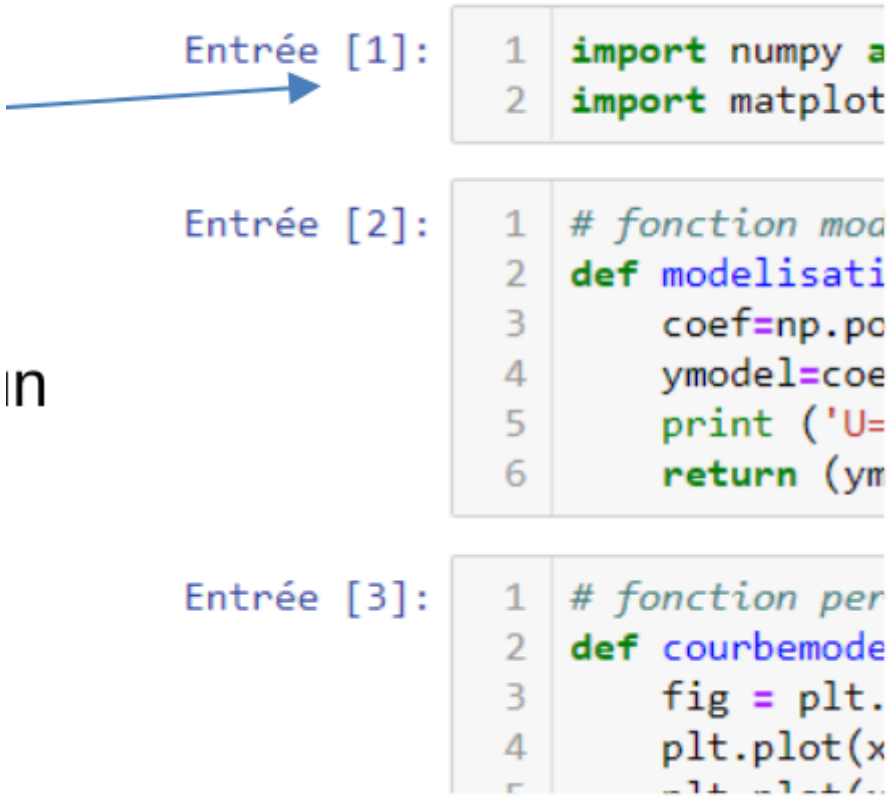



Fig. 30 – Numéro d'ordre d'exécution des cellules

Il est très important d'exécuter les cellules de code dans l'ordre du programme. Les nombres entre crochets peuvent

ne pas se suivre un par un notamment si vous exécutez une même cellule plusieurs fois de suite mais l'ordre de ces nombres doit être croissant au fur et à mesure que l'on avance dans le fichier Notebook.

Une fois exécutée, le résultat de l'exécution (sortie/output) s'affiche dans la cellule sous le code (cela peut être un message d'erreur si votre code est erroné). Il peut ne rien s'afficher si le code ne le demande pas.

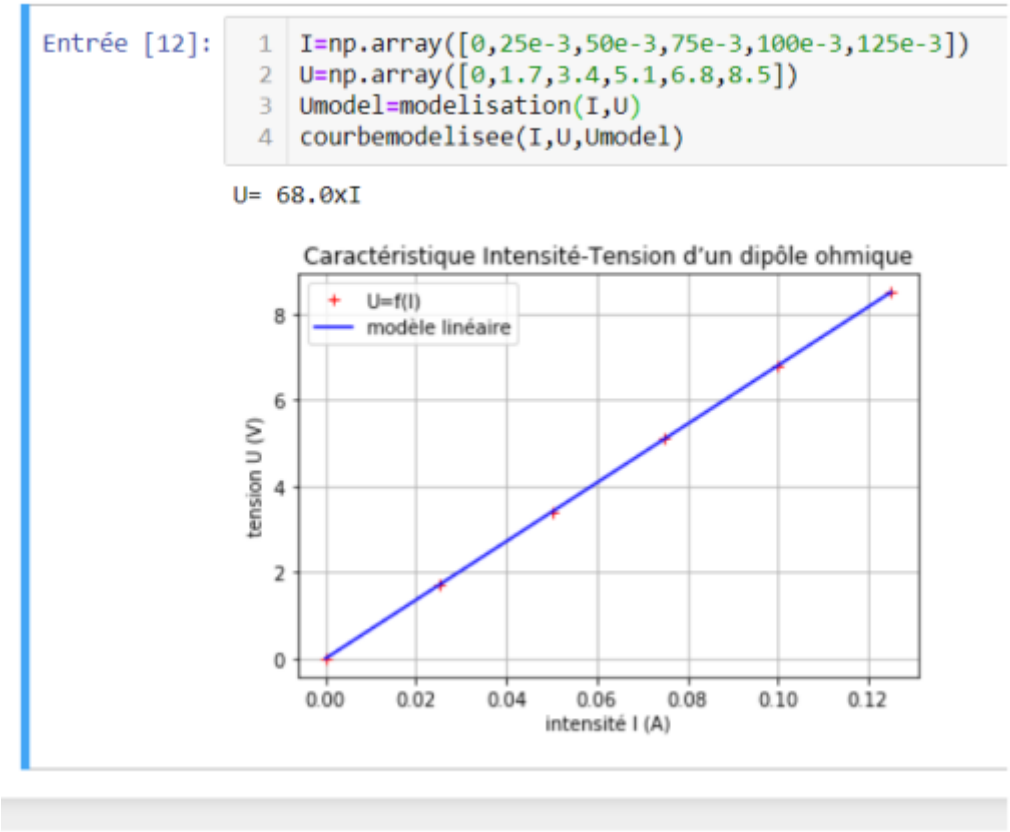



Fig. 31 – Exemple de sortie d'exécution

Il est possible d'exécuter une même cellule plusieurs fois de suite lorsque vous voulez modifier et tester le code qu'elle contient. Dans ce cas, il n'est pas nécessaire de réexécuter les cellules précédentes, ni de supprimer la sortie précédente de la cellule concernée car elle sera remplacée automatiquement lors de la nouvelle exécution de la cellule.

Enfin, il est possible de réexécuter le fichier Notebook depuis la première cellule après une ou plusieurs exécutions.

Cliquer alors sur les commandes « interrompre le noyau » puis « redémarrer le noyau (avec confirmation) » ou « redémarrer le noyau et ré-exécuter tout le notebook (sans confirmation) »

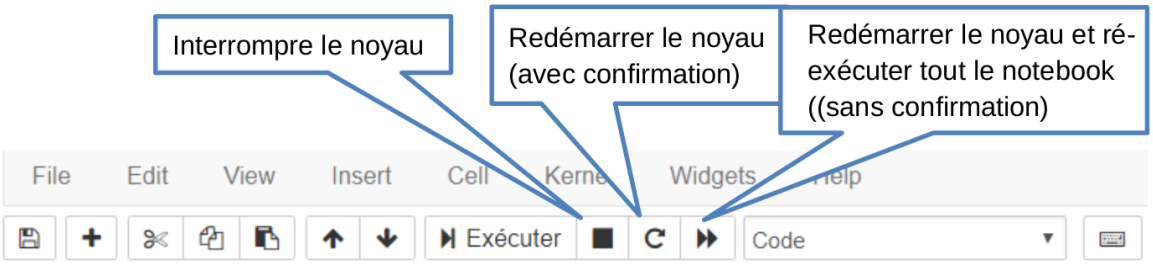



Fig. 32 – Relancer ou interrompre le noyau

Attention : « Interrompre le noyau » annule l'exécution de toutes les cellules du fichier Notebook (même si les nombres entre crochets ne s'effacent pas). Il faut donc recommencer l'exécution des cellules dès le début du fichier.

### 1.3.2 Comment définir le type de cellule ? (Markdown ou Code)

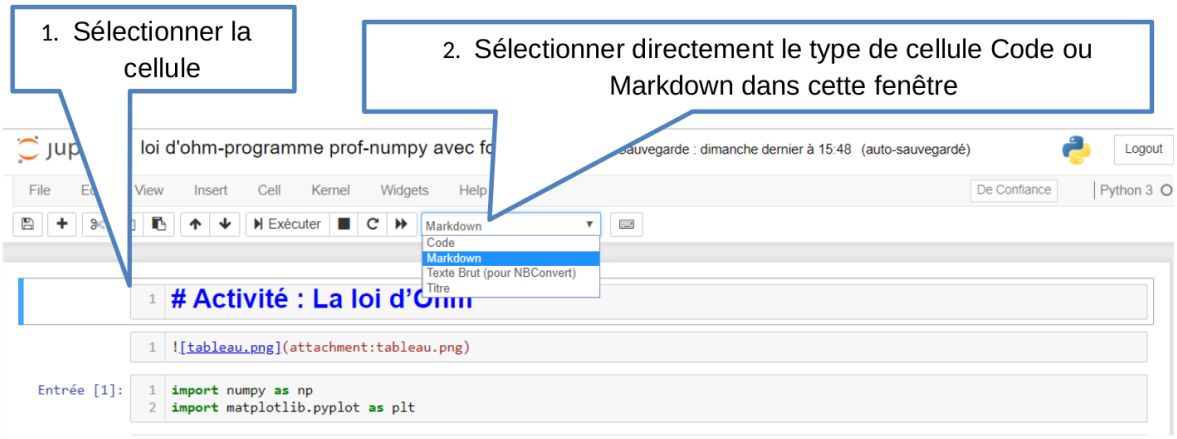



Fig. 33 – Définir le type de cellule

### 1.3.3 Comment numéroter les lignes d'une cellule ?

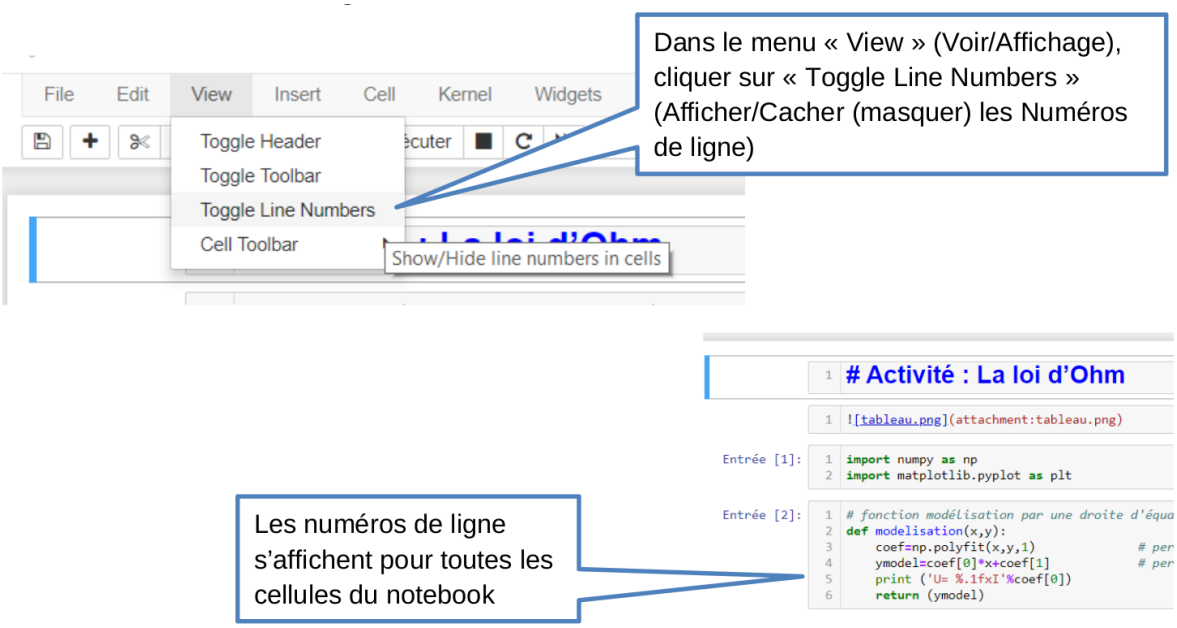



Fig. 34 – Numéroter les lignes

### 1.3.4 Comment effacer la sortie d'une seule cellule ?

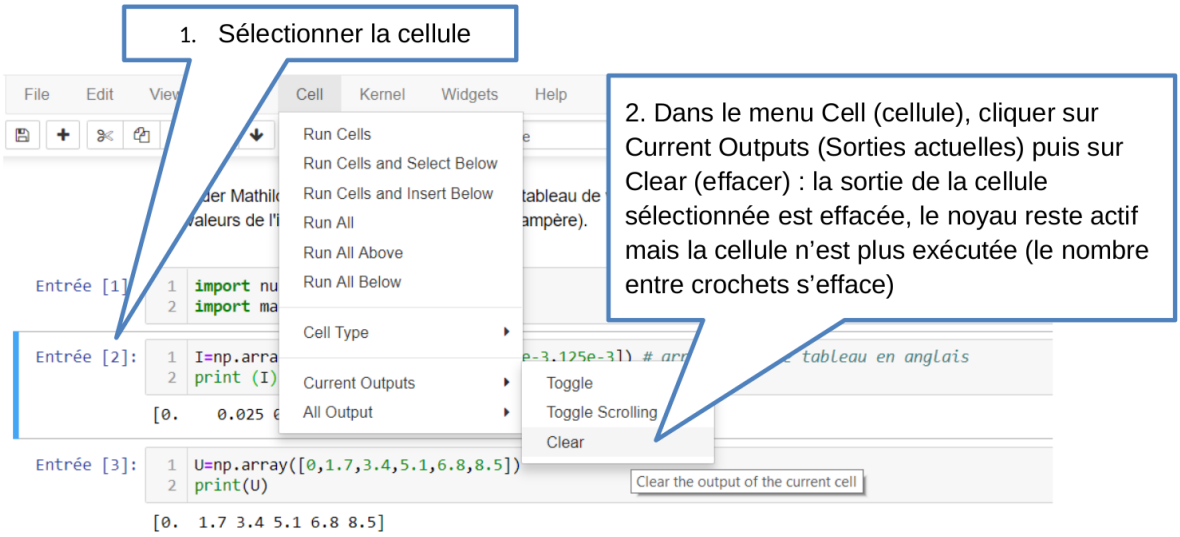



Fig. 35 – Effacer la sortie d'une cellule

### 1.3.5 Comment effacer les sorties de toutes les cellules ?

Pour "réinitialiser" le fichier notebook :

Première méthode : Restart & Clear Output

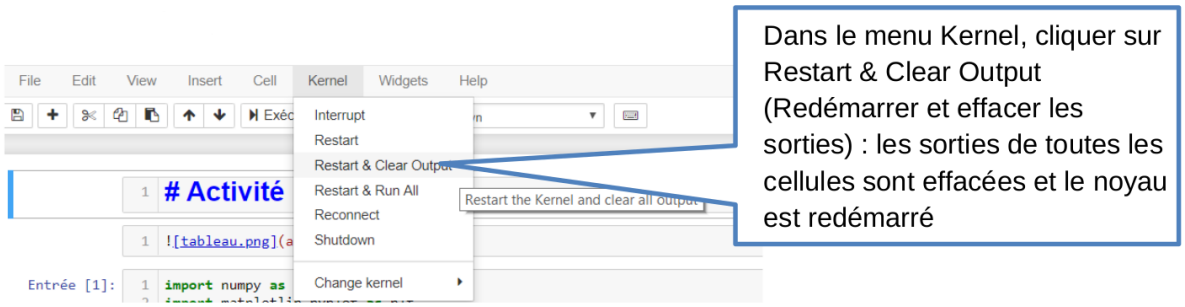



Fig. 36 – Restart and clear output

Deuxième méthode : Interrompre le noyau et All output - Clear

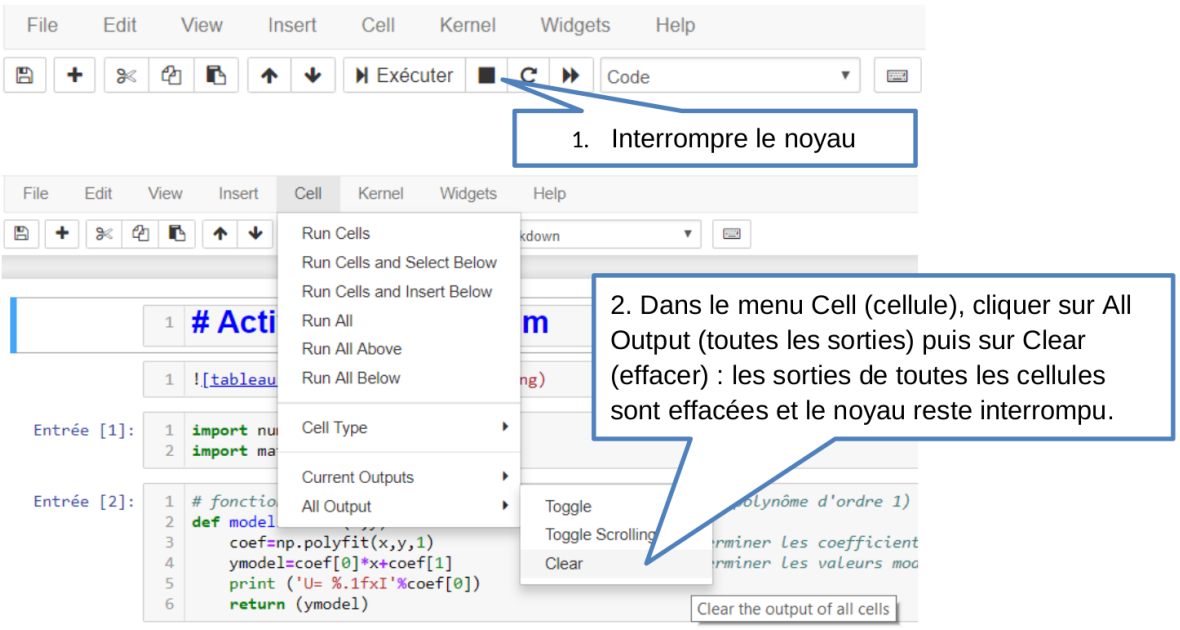



Fig. 37 – Stop kernel and all output clear

## 1.4 Quelques exemples de programmes

Avant d'aborder plus en détail la programmation avec le langage python, voici trois exemples :
- le premier calcule la structure d'un atome à partir du numéro atomique et du nombre de masse
- le second trace une courbe de décroissance radioactive par simulation de lancés de dés
- le troisième propose une animation sur les ondes progressives.

Vous pouvez télécharger et exécuter chaque notebook.

### 1.4.1 Structure de l'atome

(code sous licence creative commun CC BY-NC-SA BY Alexis Dendiével)

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

A partir du numéro atomique et du nombre de masse, ce programme :
- donne le nom et le symbole de l'atome
- précise le nombre de protons, neutrons, et électrons
- calcule la masse (approchée) du noyau et de l'atome entier
- donne la répartition électronique par couche (pour les 18 premiers éléments)
- donne la structure électronique selon la règle de Klechkowski (les exeptions n'ont pas été traitées)

Sans entrer dans la compréhension détaillée du code, je vous propose d'exécuter le programme suivant (mais vous pouvez quand même lire les commentaires :) :

[1]:
```
# fonction affichant la structure de l'atome
def structure(A, Z):
    #données
    # la masse des protons et des neutrons est approchée:
    masse_nucleon = 1.67e-27
    # masse de l'électron
    masse_electron = 9.109e-31
```

(continues on next page)

(suite de la page précédente)

```
    # liste des orbitales atomiques
    liste_orbitales = ( (1, 's', 2),  (2, 's', 2),  (2, 'p', 6),
                        (3, 's', 2),  (3, 'p', 6),  (4, 's', 2),
                        (3, 'd', 10), (4, 'p', 6),  (5, 's', 2),
                        (4, 'd', 10), (5, 'p', 6),  (6, 's', 2),
                        (4, 'f', 14), (5, 'd', 10), (6, 'p', 6),
                        (7, 's', 2),  (5, 'f', 14), (6, 'd', 10),
                        (7, 'p', 6) )
    # liste des éléments chimiques de la classification
    elements = (
    "Hydrogène H", "Hélium He", "Lithium Li", "Béryllium Be",
    "Bore B", "Carbone C", "Azote N", "Oxygène O",
    "Fluor F", "Néon Ne", "Sodium Na", "Magnésium Mg",
    "Aluminium Al", "Silicium Si", "Phosphore P",
    "Soufre S", "Chlore Cl", "Argon Ar", "Potassium K",
    "Calcium Ca", "Scandium Sc", "Titane Ti", "Vanadium V",
    "Chrome Cr", "Manganèse Mn", "Fer Fe", "Cobalt Co",
    "Nickel Ni", "Cuivre Cu", "Zinc Zn", "Gallium Ga",
    "Germanium Ge", "Arsenic As", "Sélénium Se",
    "Brome Br", "Krypton Kr", "Rubidium Rb",
    "Strontium Sr", "Yttrium Y", "Zirconium Zr",
    "Niobium Nb", "Molybdène Mo", "Technétium Tc",
    "Ruthénium Ru", "Rhodium Rh", "Palladium Pd",
    "Argent Ag", "Cadmium Cd", "Indium In", "Étain Sn",
    "Antimoine Sb", "Tellure Te", "Iode I", "Xénon Xe",
    "Césium Cs", "Baryum Ba", "Lanthane La", "Cérium Ce",
    "Praséodyme Pr", "Néodyme Nd", "Prométhium Pm",
    "Samarium Sm", "Europium Eu", "Gadolinium Gd",
    "Terbium Tb", "Dysprosium Dy", "Holmium Ho",
    "Erbium Er", "Thulium Tm", "Ytterbium Yb",
    "Lutécium Lu", "Hafnium Hf", "Tantale Ta",
    "Tungstène W", "Rhénium Re", "Osmium Os",
    "Iridium Ir", "Platine Pt", "Or Au", "Mercure Hg",
    "Thallium Tl", "Plomb Pb", "Bismuth Bi", "Polonium Po",
    "Astate At", "Radon Rn", "Francium Fr", "Radium Ra",
    "Actinium Ac",  "Thorium Th", "Protactinium Pa",
    "Uranium U", "Neptunium Np", "Plutonium Pu",
    "Américium Am", "Curium Cm", "Berkélium Bk",
    "Californium Cf", "Einsteinium Es", "Fermium Fm",
    "Mendélévium Md", "Nobélium No", "Lawrencium Lr",
    "Rutherfordium Rf", "Dubnium Db", "Seaborgium Sg",
    "Bohrium Bh", "Hassium Hs", "Meitnérium Mt",
    "Darmstadtium Ds", "Roentgenium Rg", "Ununbium Uub",
    "Ununtrium Uut", "Ununquadium Uuq", "Ununpentium Uup",
    "Ununhexium Uuh", "Ununseptium Uus", "Ununoctium Uuo")


    #calcul des masses
    masse_noyau = A * masse_nucleon
    masse = A * masse_nucleon + Z * masse_electron

    # calcul des couches électroniques

    if Z <= 2:
        couches = "(K)" + str(Z)
    elif Z <=10:
        couches = "(K)" + str(2) + "(L)" + str(Z-2)
    elif Z <= 18:
        couches = "(K)" + str(2) + "(L)" + str(8) \
                        + "(M)" + str(Z-10)
    else:
```

(continues on next page)

(suite de la page précédente)

```
        couches = "\nce calcul est limité à des numéros " + \
                  "atomiques inférieurs ou égal à 18"


    # calcul des orbitales atomiques
    orbitale = 0
    n_restant = Z
    structure = ""
    while n_restant > 0:
        (n, nom, ne) = liste_orbitales [orbitale]
        if n_restant < ne:
            nmin = n_restant
        else:
            nmin = ne
        structure = structure + str(n) + nom + str(nmin) + ' '
        n_restant = n_restant - nmin
        orbitale = orbitale + 1

    # impression des résultats
    print("----------------------")
    print ("RESULTATS pour A =", A, " et Z =",Z)
    # impression de l'élément et de son symbole
    print("il s'agit de l'élément :", elements[Z-1])
    # impression de la structure de l'atome
    print("nombre de protons : ", Z )
    print("nombre de neutrons : ", A - Z)
    print("nombre d'électrons : ", Z)
    # impression des masses, du noyau et de l'atome
    print("masse du noyau : ",
          "{0:.3e}".format(masse_noyau), ' kg')
    print("masse de l'atome : ",
          "{0:.3e}".format(masse), ' kg')
    print("")
    # impression des couches et structure électronique
    print("remplissage des couches électroniques :",
          couches)
    print("la structure électronique est :",
          structure)
```

[2]:
```
# Utilisation : calcul de la structure
# Chlore
structure(A=36,Z=17)
# Carbone
structure(A=12,Z=6)
```

```
--------------
RESULTATS pour A = 36  et Z = 17
il s'agit de l'élément : Chlore Cl
nombre de protons :  17
nombre de neutrons :  19
nombre d'électrons :  17
masse du noyau :  6.012e-26  kg
masse de l'atome :  6.014e-26  kg

remplissage des couches électroniques : (K)2(L)8(M)7
la structure électronique est : 1s2 2s2 2p6 3s2 3p5
--------------
RESULTATS pour A = 12  et Z = 6
il s'agit de l'élément : Carbone C
nombre de protons :  6
nombre de neutrons :  6
```

(continues on next page)

(suite de la page précédente)

```
nombre d'électrons :  6
masse du noyau :  2.004e-26  kg
masse de l'atome :  2.005e-26  kg

remplissage des couches électroniques : (K)2(L)4
la structure électronique est : 1s2 2s2 2p2
```

### 1.4.2 Simulation d'une décroissance radioactive



Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

Simulons une décroissance radioactive par un lancer de dès :

— Un dé à 6 faces représente un atome radioactif.
— Il se désintègre quand, lors d'un lancé, il indique 6.

Sur un échantillon de plusieurs dés, la simulation consiste à retirer à chaque lancé les dés ayant indiqué 6. Nous traçons ensuite la courbe donnant le nombre d'atomes radioactifs en fonction du numéro de lancé.

Ici encore, vous êtes juste invités à excécuter le programme.

[1]:
```
# programme de simulation de décroissance radioactive
# par le jet de dés

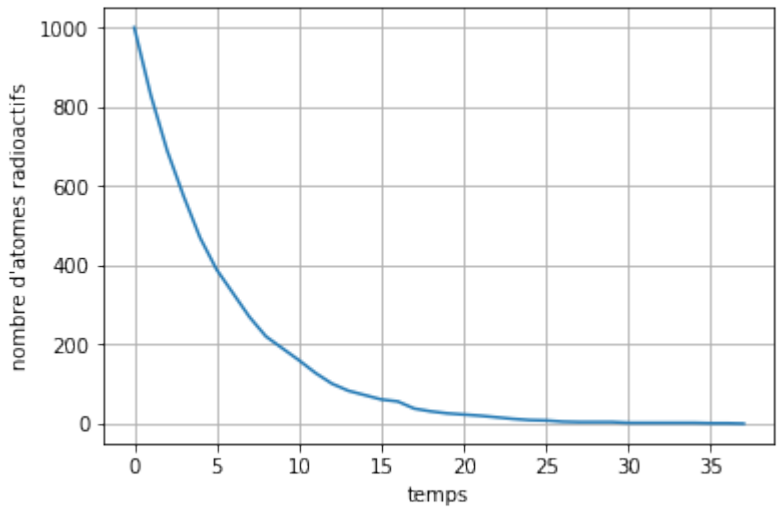
# importations de fonctions utiles
import matplotlib.pyplot as plt
%matplotlib inline
from random import randint

# fonction permettant de lancer la simulation pour un
# nombre n d'atomes
def simulation(n):
    # initialisation des données
    nombrelance = 0
    temps = [0]
    radioactifs = [n]

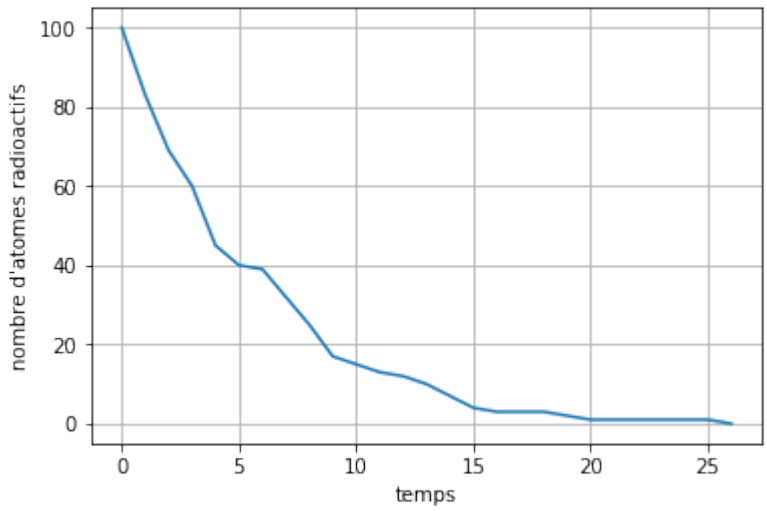
    # coeur du programme
    while n > 0:
            desintegration = 0
            for i in range(n):
                tirage = randint(1,6)
                if tirage == 6:
                    desintegration = desintegration + 1
            n = n - desintegration
            nombrelance = nombrelance + 1
            temps.append(nombrelance)
            radioactifs.append(n)

    # affichage
    plt.figure()
    plt.plot(temps, radioactifs)
    plt.grid()
    plt.xlabel("temps")
    plt.ylabel("nombre d'atomes radioactifs")
    plt.show()
    plt.close()
```

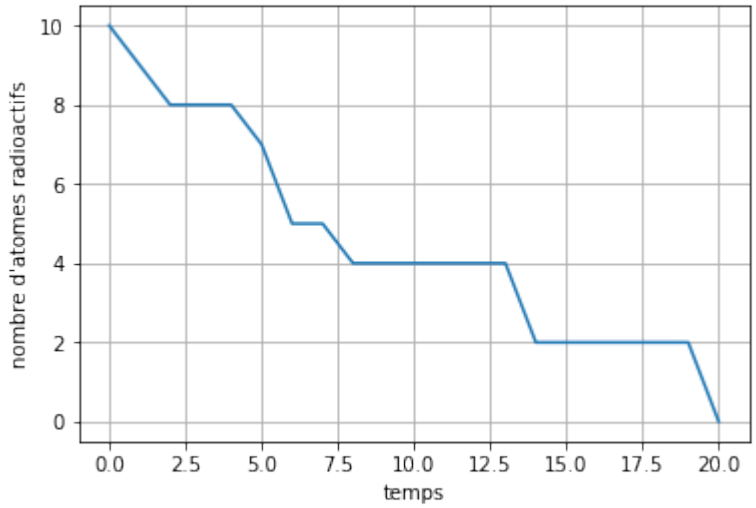
[2]:
```
# utilisation
simulation(n=1000)
```




[3]:
```
simulation(n=100)
```




[4]:
```
simulation(n=10)
```




### 1.4.3 Animation pour une onde progressive

(code sous licence creative commun CC BY-NC-SA BY Gaëlle Rebolini)

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Ce programme propose l'animation d'une onde progressive à partir des valeurs :
- de l'amplitude
- de la période
- de la longueur d'onde

(Une fois lancée, patientez un peu pour voir l'animation)

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import animation, rc

# Fonction permettant l'animation d'une onde
# d'amplitude Ymax (en m), de période T (en secondes)
# et de longueur d'onde lamb (en m). L'affichage sur x
# se fait entre 0 et xmax (xmax = 3 longueurs d'onde
# si le paramètre n'est pas fourni)
def onde_progressive(Ymax, T, lamb, xmax=None):
    print("Calcul de l'animation en cours, "
          "merci de patienter...")
    xmin=0
    if xmax is None:
        xmax=3*lamb
    nbx=100

    fig=plt.figure(figsize=(12,10))
    line = plt.plot([], [],'bo-')
    plt.xlim(xmin,xmax)
    plt.ylim(-Ymax,Ymax)
    plt.grid()
    plt.xlabel("x(m)")
    plt.ylabel("y(m)")
    plt.title("animation : propagation d'une "
```

(continues on next page)

(suite de la page précédente)

```
                "onde le long d'une corde")

    def init():
        line[0].set_data([], [])
        return (line)

    def animate(i):
        dt=0.03
        t=i*dt
        x = np.linspace(xmin, xmax, nbx)
        y = Ymax*np.cos(2 * np.pi * (x/lamb - t/T))
        line[0].set_data(x, y)
        return (line)

    anim = animation.FuncAnimation(
        fig,
        animate,
        init_func=init,frames=50,
        interval=30,
        blit=True,
        repeat=False)

    plt.close()

    # lignes de code à remplacer par plt.show()
    # sur un éditeur python (spyder...)
    rc('animation', html='jshtml')
    print("Calcul terminé, affichage en cours de téléchargement...")
    return anim

# patience, c'est un peu long à s'afficher...
```

[2]:
```
# Utilisation, animation d'une onde d'amplitude
# Ymax=0,2m, de période T=1s et de longueur d'onde
# lamb=0,4m, pour un affichage jusqu'à xmax=2m
onde_progressive(Ymax=0.2, T=1, lamb=0.4, xmax=2)
```

```
Calcul de l'animation en cours, merci de patienter...
Calcul terminé, affichage en cours de téléchargement...
```

[2]:
```
<matplotlib.animation.FuncAnimation at 0x7fd9fe36ee80>
```

[3]:
```
# Utilisation, animation d'une onde d'amplitude
# Ymax=0,2m, de période T=1s et de longueur d'onde
# lamb=0,9m, pour un affichage jusqu'à xmax=2m
onde_progressive(Ymax=0.2, T=1, lamb=0.9, xmax=2)
```

```
Calcul de l'animation en cours, merci de patienter...
Calcul terminé, affichage en cours de téléchargement...
```

[3]:
```
<matplotlib.animation.FuncAnimation at 0x7fd9fe0202b0>
```

CHAPITRE 2

# Les bases, mémos et activités

## 2.1 Les bases de la programmation python

Cette section présente quelques bases pour la programmation en python, avec un accent particulier en direction d'une utilisation en sciences physiques. D'autres ressources sont disponibles en ligne pour un apprentissage du python plus « généraliste ».

### 2.1.1 La structure d'un programme



`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

#### 2.1.1.1 Structure général d'un programme

Elle correspond pour un programme simple au déroulé suivant

- importation des bibliothèques
- définition des constantes, des fonctions
- appel des entrées
- corps de programme
- affichage des résultats.

#### 2.1.1.2 L'importance du commentaire

Les commentaires dans un programme sont d'une importance capitale. ils servent : - à l'auteur du programme - au lecteur du programme - à celui qui souhaite modifier le programme.

Sans aucun impact sur l'execusion, ils gagnent à être le plus précis possible.

[1]:
```
# ceci est un commentaire de programme
```

Tout ce qui est précédé du signe # dans un programme est un commentaire.

Voyons ceci sur un exemple :

le programme suivant, calculant la force de gravitation entre deux masses m1 et m2 séparées d'une distance d, est donnée en version brute et en version commentée :

[2]:
```
G = 6.67e-11
print("nous nous nous proposons de calculer la force de gravitation "
      "s'exerçant entre deux masses m1 et m2 séparées d'une distance d")
# Masse m1 en kg
m1 = 50
# Masse m2 en kg
m2 = 70.5
# Distance d en mètres
d = 2.5
FG = G*m1*m2/d**2
print("masse m1 : {0:.2e} kg".format(m1))
print("masse m2 : {0:.2e} kg".format(m2))
print("masse d : {0:.2e} m".format(d))
print("==> la valeur de la force de gravitation est: {0:.2e} N".format(FG))
```

```
nous nous nous proposons de calculer la force de gravitation s'exerçant entre deux␣
↪masses m1 et m2 séparées d'une distance d
masse m1 : 5.00e+01 kg
masse m2 : 7.05e+01 kg
masse d : 2.50e+00 m
==> la valeur de la force de gravitation est: 3.76e-08 N
```

Nous voyons dans cet exemple simple que la lecture du programme est beaucoup plus aisée avec les commentaires. De plus, le saut d'une ligne n'a aucun impact sur la programmation, et permet une plus grande lisibilité.

Il est donc fortement recommandée de prendre le temps de commenter ses programmes.

(Il en sera d'ailleurs de même pour les fonctions)

#### 2.1.1.3 Les noms de variable

Rien ni personne n'oblige à utiliser tel ou tel nom de variable dans python. La variable n'a pas à être défini au début du programme, elle pointe sur un objet et en présente les caractéristiques ; en revanche, il est obligatoire d'avoir assigné une valeur à une variable **avant** de l'utiliser.

Il est cependant fortement recommandé d'utiliser des noms de variables donnant sens au programme, pour plus de clarté envers les personnes susceptibles de le lire ou de le modifier. exemples :

- « atome » plutôt que « a »
- « forcedegravitation » plutôt que « f »
- « temps » plutôt que « t » . . .

Une variable peut être écrite en majuscule ou minuscules, voire même avec des accents, mais les bonnes pratiques incitent plus à utiliser des minuscules pour les variables, sans accents, et en séparant les mots par des « underscore »

[3]:
```
élément=12 # mal
element=12 # bien
Numéroatomique=12 # mal
numero_atomique=12 # bien
```

[4]:
```
# très mal (pas possible, en fait)
numero atomique=12
```

```
  File "<ipython-input-4-c80d00387f69>", line 2
    numero atomique=12
                   ^
SyntaxError: invalid syntax
```

#### 2.1.1.4 L'importance de l'indentation

Un programme python n'a pas de code pour le début et la fin d'une séquence d'instructions. Sa structuration suit une indentation (espaces en début de ligne) qui définit la structuration du programme.

Le « : » après les instructions tabule automatiquement la ligne suivante.

Pour une instruction else suivant un if par exemple, il faut revenir en arrière afin que la structure soit correcte et lisible.

[5]:
```
n = 3
if n>5:
    print("n est plus grand que 5")
else:
    print("n est plus petit ou égal à 5")
```

```
n est plus petit ou égal à 5
```

Cette règle d'écriture structure le programme et demande de la rigueur. Il est par ailleurs fortement conseillé de ne pas faire des lignes de code trop grandes. Si celle-ci est importante, un passage à la ligne en milieu d'instruction donnera davantage de lisibilité au programme.

#### 2.1.1.5 Quelques bibliothèques en python

Même si la bibliothèque standard python est bien développée, certaines bibliothèques sont bien utiles : ### numpy C'est une bibliothèque très utile pour les calculs numériques.

[6]:
```
# importation de la bibliothèque numpy
import numpy as np
```

La bibliothèque de calcul, dont vous trouverez facilement les utilisations sur le net, est ici importée en début de programme sous le nom np.

[7]:
```
# quelques exemples de calcul
print(np.pi)
print (np.sin(np.pi/4))
print (np.cos(np.pi/4))
```

```
3.141592653589793
0.7071067811865475
0.7071067811865476
```

##### 2.1.1.5.1 matplotlib.pyplot

C'est une bibliothèque permettant de faire facilement des graphiques.

[8]:
```
# programme de tracé simple avec des valeurs d'abscisse et d'ordonnée

# importation de la bibliothèque matplotlib.pyplot
import matplotlib.pyplot as plt
%matplotlib inline

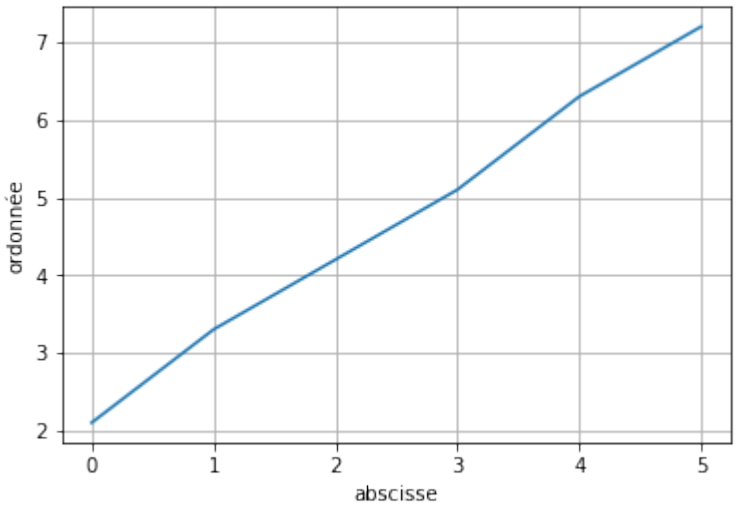
# initialisation des données
x = (0,1,2,3,4,5)
y = (2.1, 3.3, 4.2, 5.1, 6.3, 7.2)

# affichage
plt.plot(x, y)
plt.grid()
plt.xlabel("abscisse")
```

(continues on next page)

(suite de la page précédente)

```
plt.ylabel("ordonnée")
plt.show()
plt.close()
```




#### 2.1.1.5.2 math

C'est une bibliothèque qui permet de faire facilement les calculs mathématiques

[9]:
```
# importation de la bibliothèque math
import math as math

# quelques exemples
print (math.cos(math.pi))
print (math.sqrt(5))
print ("la valeur approchée du nombre d'or est", (1+math.sqrt(5))/2)
```

```
-1.0
2.23606797749979
la valeur approchée du nombre d'or est 1.618033988749895
```

#### 2.1.1.5.3 random

C'est une bibliothèque permettant de créer des nombres aléatoires

[10]:
```
# importation de la bibliothèque random
import random as rd

# impression de 5 nombres aléatoires compris entre 1 et 100
for i in range (5):
    print(rd.randint(1,100))
```

```
100
83
40
11
27
```

#### 2.1.1.5.4 turtle

Cette bibliothèque permet de tracer des graphiques en suivant la « tortue ». (Cette bibliothèque & cet exemple ne fonctionnent pas sur un environnement jupyter en ligne)

[ ]:
```
# importation de la bibliothèque
import turtle as tu

tu.reset()
tu.speed(1)
for i in range(6):
  tu.fd(100)
  tu.rt(360/6)
```

liste non exaustive des fonctions turtle

- fd(n) avance de n
- bk(n) recule de n
- rt(n) tourne à droite de n degrés
- lt(n) tourne à gauche de n degrés
- clear() efface l'écran
- penup() lève le stylo
- pendown() baisse le stylo
- reset() efface l'écran, remet la tortue au centre et réinitialise ses paramètres
- showturtle() montre la tortue
- hideturtle() cache la tortue
- speed(n) Change la vitesse de 1(lent) à 10 (rapide). La valeur spéciale 0 est la plus rapide.
- tracer(n,d)
- update() Force l'affichage des graphismes en attente
- bye() Referme la fenêtre
- setup(w,h) Ouvre une fenêtre de taille wxh

### 2.1.1.6 Conclusion

Ce notebook donne les premiers conseils afin de structurer un programme python. Ce qu'il faut retenir, c'est la recherche de la clarté tant dans le code que dans les commentaires.

## 2.1.2 Affectation des variables et impression des résultats



Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

### 2.1.2.1 Les differents types de variables

Python est un langage objet, l'affectation du type de variable se fait par l'objet, il n'est donc pas nécessaire de déclarer le type d'une variable.

Il existe plusieurs types d'objets : - entier - flottant - chaine de caractère - booléen ...

voyons par l'exemple l'affectation des variables :

[1]:
```
n = 1
type (n)
```

[1]:
```
int
```

La commande type renvoi le type de variable. Ici l'objet 1 étant un entier, la variable n prend le type de l'objet.

```
[2]: n = "1"
     type(n)
[2]: str
```

l'objet « 1 » étant une chaîne de caractère, la variable n prend le type chaine de caractère (str)

```
[3]: n = 1.13e-7
     type (n)
[3]: float
```

de même ici, 1.13e-7 est un flottant, la variable n prend le type flottant.

```
[4]: n = (1, 2, 3, 5, 7, 11, 13, 17, 19)
     type (n)
[4]: tuple
```

En conclusion, en python, c'est l'objet qui détermine le type de la variable.

#### 2.1.2.2 La fonction input

##### 2.1.2.2.1 Présentation

**Attention, l'utilisation de ‘‘input‘‘ est fortement déconseillée, il vaut mieux utiliser des fonctions, voir un peu après**.

Quand on demande à l'utilisateur d'entrer une grandeur, on utilise la commande input. Il faut cependant faire attention au type de grandeur :

```
[5]: n = input ("entrer un nombre entier : ")
     type(n)
     entrer un nombre entier : 42
[5]: str
```

Par défaut, la fonction input entre une chaîne de caractère, avec laquelle nous ne pouvons pas faire de calculs. Il faut alors préciser que nous souhaitons une valeur entière :

```
[6]: n = int(input("entrer un nombre entier : "))
     type(n)
     entrer un nombre entier : 42
[6]: int
```

```
[7]: n = float(input("entrer un nombre : "))
     print(n)
     entrer un nombre : 42.42
     42.42
```

Ici n est un nombre réel.

##### 2.1.2.2.2 Comment ne pas utiliser `input`

Imaginons un programme devant calculer le nombre de neutrons d'un atome ; nous avons besoin de `A` et de `Z` pour faire le calcul.

Une (mauvaise) solution serait d'écrire le programme de la manière suivante :

```
A = int(input("A ?"))
Z = int(input("Z ?"))
print("Nombre de protons : ", A-Z)
```

Ce code nécessite une intéraction avec l'utilisateur pour être utilisé, on ne peut pas l'utiliser de manière automatique pour calculer le nombre de protons d'un ensemble d'atomes => pas bien :)

La « bonne » solution est d'écrire l'action que l'on souhaite faire sous la forme d'une fonction (voir un peu plus loin pour les détails sur les fonctions), en détaillant et documentant les arguments à utiliser, et de faire utiliser cette fonction à l'utilisateur.

[8]:
```
def calcul_nombre_protons(A, Z):
    """
    Calcul et affichage du nombre de neutrons d'un atome

    :param A: nombre de masse
    :param Z: numéro atomique
    """
    print("Nombre de neutrons : ", A-Z)


calcul_nombre_protons(A=12, Z=6)
```

```
Nombre de neutrons :  6
```

#### 2.1.2.3 Print et l'écriture scientifique

Quand on souhaite imprimer un résultat à l'écran, on peut utiliser la fonction print.

Cependant, dans un notebook jupyter, il n'est pas toujours utile d'utiliser print : en effet, la valeur retournée par la dernière instruction est automatiquement affichée comme résultat de la cellule.

[9]:
```
# Affichage sans "print" (notebook jupyter)
n = 10
m = n/7
m
```

[9]:
```
1.4285714285714286
```

[10]:
```
# Affichage avec print
n = 10
m = n/7
print(m)
```

```
1.4285714285714286
```

L'instruction print a cependant quelques avantages : il est par exemple souvent utile en sciences de présenter les résultats avec un nombre correct de chiffres significatifs.

[11]:
```
print("{0:.2e}".format(m))
```

```
1.43e+00
```

Ce formalisme dans le print permet de présenter les résultats avec le nombre de chiffres significatifs corrects. Nous avons ici l'écriture scientifique à trois chiffres significatifs.

La commande print permet d'afficher des caractères afin de présenter le résultat :

[12]:
```
print("le résultats du calcul est: ", "{0:.2e}".format(m))
```

```
le résultats du calcul est:  1.43e+00
```

On remarque que les chaînes de caractères sont entre » « . Nous pouvons également utiliser " ". Il faut cependant faire attention, car si la chaîne de caractère contient un ", elle doit être entourée de » » :

[13]:
```
print("ceci est une écriture correcte")
print('ceci est une écriture équivalente')
print("l'usage de l'apostrophe nécessite ce formalisme")
```

```
ceci est une écriture correcte
ceci est une écriture équivalente
l'usage de l'apostrophe nécessite ce formalisme
```

#### 2.1.2.3.1 Alignement des résultats

il peut être utlise d'aligner les résultats pour une lecture facilitée.

L'instruction print(« { :60} ».format()) calera ce qu'il y a dans la parenthèse du format sur 60 caractères, et permet donc cet alignement.

Exemple :

[14]:
```
n = 10
m = n/7
print('{:60}'.format("le résultat avec un chiffre significatif est: "),
      "{0:.0e}".format(m))
print('{:60}'.format("le résultat avec deux chiffres significatif est: "),
      "{0:.1e}".format(m))
print('{:60}'.format("le résultat avec trois chiffres significatif est: "),
      "{0:.2e}".format(m))
print('{:60}'.format("le résultat avec quatre chiffres significatif est: "),
      "{0:.3e}".format(m))
```

```
le résultat avec un chiffre significatif est:                1e+00
le résultat avec deux chiffres significatif est:             1.4e+00
le résultat avec trois chiffres significatif est:            1.43e+00
le résultat avec quatre chiffres significatif est:           1.429e+00
```

Il est possible également d'utiliser la tabulation, grâce à la commande `\t`

[1]:
```
n = 10
m = n/7
print("le résultat avec deux chiffres significatifs est: \t",
      "{0:.1e}".format(m))
print("le résultat avec trois chiffres significatifs est: \t",
      "{0:.2e}".format(m))
print("le résultat avec trois chiffres significatifs est: \t\t",
      "{0:.2e}".format(m))
print("le résultat avec trois chiffres significatifs est: \t\t\t",
      "{0:.2e}".format(m))
```

```
le résultat avec deux chiffres significatifs est:  1.4e+00
le résultat avec trois chiffres significatifs est:  1.43e+00
le résultat avec trois chiffres significatifs est:  1.43e+00
le résultat avec trois chiffres significatifs est:  1.43e+00
```

#### 2.1.2.3.2 Le passage à la ligne

Il peut être intéressant dans la présentation des résultats d'empécher le passage à la ligne. On utilise pour cela la commande end=" "

[16]:
```
print("premier résultat", end =" ")
print("second résultat")
```

```
premier résultat second résultat
```

La commande end = " " permet également de pouvoir positionner des séparateurs.

[17]:
```
for i in range (10):
    print(i, end=" - ")
```
```
0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 -
```

Il peut également être intéressant de passer volontairement à la ligne. On utilise pour cela la commande `\n`

[18]:
```
print("ceci est le début d'une phrase \nceci est la suite à la ligne")
```
```
ceci est le début d'une phrase
ceci est la suite à la ligne
```

#### 2.1.2.3.3 Affichage de caractères spéciaux

Il peut s'avérer nécessaire d'utiliser des caractères spéciaux. On utilise pour cela la commande

[19]:
```
print("le mot \"Pierre\" est entre guillements")
```
```
le mot "Pierre" est entre guillements
```

#### 2.1.2.3.4 Séparateurs

Enfin, il peut être utlise d'utiliser des séparateurs. On utilise pour cela la commande sep :

[20]:
```
print("premier résultat",
      "deuxième résultat",
      "troisième résultat",
      ".",
      sep = " -- ")
```
```
premier résultat -- deuxième résultat -- troisième résultat -- .
```

### 2.1.2.4 conclusion

Nous avons vu l'essentiel des fonctions imput et print. Il existe d'autres méthodes que l'on trouvera facilement quand l'usage s'en fera sentir.

## 2.1.3 Les fonctions



Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

Lorsqu'on écrit un programme, on a besoin de fonctions diverses et variées.

- Les fonctions mathématiques arithmétiques (multiplication, addition,...) ou logiques (OU, ET, ...) classiques sont accessibles dans la bibliothèque de « base » de python et ne recquièrent aucune ligne de code supplémentaire pour y avoir accès.
- Les fonctions mathématiques plus sophistiquées sont inclues dans des **bibliothèques** spécifiques disponibles qu'il faut importer avant de pouvoir utiliser ces fonctions.
- On peut aussi avoir besoin de créer ses propres fonctions personnalisées.

**Une bibliothèque est donc un ensemble de fonctions prédéfinies.**

#### 2.1.3.1 Bibliothèques externes disponibles

Une des grandes forces du langage Python réside dans le nombre important de bibliothèques logicielles externes disponibles. Celles-ci sont mises à disposition afin de pouvoir être utilisées sans avoir à les réécrire.

Quelques exemples :

— la librairie *math* contient entre autres les fonctions trigonométriques, la racine carrée, les recherches de PGCD, les factorielles ...
— la librairie *random* permet d'avoir accès à de nombreuses fonctions en rapport avec la génération de nombres aléatoires,
— la librairie *matplotlib* contient toutes les fonctions permettant de générer et de gérer l'affichage de graphiques,
— la librairie *numpy* contient de nombreux outils mathématiques (trigonométriques, tableaux,...) et permet entre autres de modéliser des ensembles de valeurs.

1. L'importation des bibliothèques doit se faire en tête de programme
2. Pour importer une bibliothèque, il suffit d'écrire la ligne : **import nom_de_la_bibliothèque**
3. L'accès à la fonction s'effectue ainsi : **nom_de_la_bibliothèque.nom_de_la_fonction**
4. On peut aussi donner un petit surnom à la bibliothèque par souci de simplification : **import nom_de_la_bibliothèque as surnom**
5. On peut aussi parfois ne vouloir importer qu'une fonction spécifique et non pas la totalité d'une bibliothèque. Il suffit alors d'écrire : **from nom_de_la_bibliothèque import fonction**. Attention dans ce cas, l'accès à la fonction s'effectue ainsi : **nom_de_la_fonction**

Quelques liens utiles...

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

https://docs.python.org/fr/3/library/random.html

[1]:
```
# Importation de la bibliothèque math sans surnom
import math
print(math.cos(math.pi))  # permet d'avoir accès à la fonction cosinus ainsi qu'à
↪la valeur de pi et d'afficher le résultat
```

```
-1.0
```

[2]:
```
# Importation de la bibliothèque random avec surnom
import random as rd
print(rd.randint(1,10)) # la fonction randint permet d'afficher à l'écran un
↪nombre aléatoire entier compris entre 1 et 10
```

```
7
```

[3]:
```
# Importation de la seule fonction randint à partir de la bibliothèque random
from random import randint
print(randint(1,10))    # ne pas écrire random.randint() mais uniquement randint()
```

```
2
```

#### 2.1.3.2 Fonctions personnalisées

Un programme écrit « linéairement » est peu lisible . On préfère en général le décomposer en plusieurs sous-programmes, nommés fonctions.

Une fonction est donc un ensemble d'instructions !

Les avantages de définir des fonctions sont multiples :

— le programme est plus lisible car architecturé, et se comprend plus facilement ;
— le code est réutilisable ;
— le programme est moins long : il suffit de définir une fonction pour effectuer une tâche précise, puis appeler cette fonction plusieurs fois si nécessaire ;
— le programme est plus facile à corriger et à améliorer.

#### 2.1.3.2.1 Création d'une fonction

1. Les fonctions sont souvent écrites en tête de programme, après les imports de bibliothèques
2. La définition d'une fonction commence toujours par la ligne **def nom_de_la_fonction() :** (ne pas oublier les : à la fin, erreur classique)
3. Les lignes de codes de cette fonction sont ensuite écrites en-dessous avec une indentation (décalage vers la droite).
4. Lorsqu'il a besoin de cette fonction, le programme principal (PP) doit l'appeler : **nom_de_la_fonction()**
5. Le PP peut avoir besoin d'échanger des informations avec la fonction.
6. Il est conseillé de documenter la fonction en utilisant la syntaxe `"""  (...)  """` juste après la déclaration de fonction

Les exemples ci-après sont progressifs. Les fonctions étudiées n'ont pas d'autre intérêt que d'illustrer les différents échanges possibles entre le PP et la fonction.

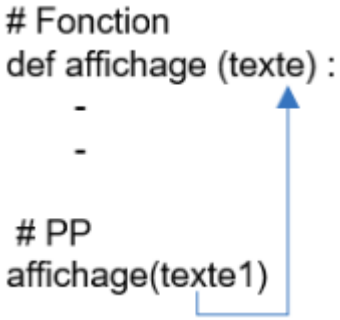
[4]:
```
#Dans ce premier exemple, il n'y a pas d'échange d'informations entre le PP et la fonction
# Définition de la fonction nommée félicitations

def félicitations() :                # déclaration de la fonction
    """
    Affiche un message de fécilicitations
    """
    print("Bien joué !")            # ligne de code de la fonction

#Programme principal
félicitations()                     # Appel de la fonction
```

```
Bien joué !
```

Dans ce second exemple, le PP envoie le texte à imprimer à la fonction. Le PP envoie donc un paramètre (ici la




variable texte1) à la fonction qui le range dans sa propre variable texte.

[5]:
```
# Définition de la fonction nommée affichage
def affichage(texte) :              # la variable texte contiendra la variable envoyée par le PP
    """
    Affichage du texte donné en argument

    :param texte: texte à afficher
    """
    print(texte)
```

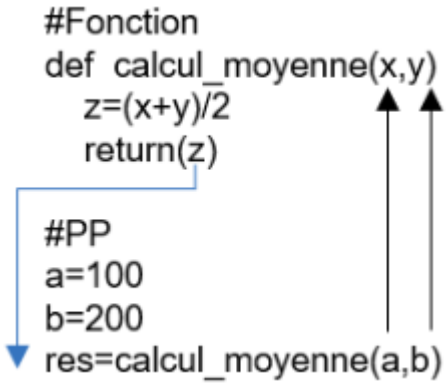
(continues on next page)

(suite de la page précédente)

```
#Programme principal
texte1 ="Bien joué !"
affichage(texte1)                    # Premier appel de la fonction avec envoi de la␣
↪variable texte1
texte2 ="Perdu !"
affichage(texte2)                    # Second appel de la fonction avec envoi de la␣
↪variable texte2
```

```
Bien joué !
Perdu !
```

Dans ce troisième exemple, le PP envoie deux paramètres (ici les variables a et b) à la fonction qui les range dans deux variables locales (ici x et y). Après avoir rempli son rôle, la fonction renvoie le résultat au PP. Celui-ci range




alors le résultat dans sa variable res.

[6]:
```
# Définition de la fonction nommée calcul_moyenne
def calcul_moyenne(x,y) :              # la variable x contiendra la valeur 100 et␣
↪y contiendra la valeur 200
    """
    Calcul de la moyenne de x et y

    :param x: valeur x
    :param y: valeur y
    :return: moyenne de x et y
    """
    z=(x+y)/2
    return(z)                          # la fonction renvoie le résultat z au PP

#Programme principal
a=100
b=200
res=calcul_moyenne(a,b)                # Appel de la fonction avec envoi de 2␣
↪variables a et b ET stockage du résultat
print(res)                             # renvoyé par la fonction dans la variable␣
↪res.
```

```
150.0
```

#### 2.1.3.2.2 Passage des arguments d'une fonction

La lecture du code et de la documentation d'une fonction est utile pour connaître l'ordre et la signification des arguments d'une fonction, cependant ce n'est pas toujours pratique au milieu d'un long programme de s'y reporter, surtout dans le cas de fonctions avec un grand nombre d'arguments.

Imaginons une fonction calculant la valeur du vecteur accélération moyen à partir des coordonnées de deux vecteurs vitesse $\vec{v_1}$ et $\vec{v_2}$ et de l'intervalle de temps $\Delta t$

On peut écrire cette fonction de plusieurs manières :

```
def calcul_acceleration(vx1, vy1, vz1, vx2, vy2, vy2, dt):
```

```
def calcul_acceleration(vx1, vx2, vy1, vy2, vz1, vz2, dt):
```

```
def calcul_acceleration(dt, vx1, vx2, vy1, vy2, vz1, vz2):
```

etc..

L'appel à la fonction en passant les arguments dans l'ordre nécessite de se rappeler quelle est la forme choisie pour l'ordre des arguments :

```
# pas très parlant...
calcul_acceleration(1, 0, 2, 3,0, 3, 0.04)
```

Heureusement, le python permet d'appeler une fonction en précisant le nom de chaque argument... et dans ce cas, il n'est plus nécessaire de donner les arguments dans l'ordre !

```
# beaucoup plus clair
calcul_acceleration(vx1=1, vy1=0, vz1=2, vx2=3, vy2=0, vz2=3, dt=0.04)
```

Mieux encore : il est possible de préciser dans la définition de la fonction des *valeurs par défaut* pour les arguments qui ne sont pas précisés lors de l'appel de la fonction.. si on travaille en 2 dimensions, par exemple, les valeurs en Z sont systématiquement nulles, on pert donc du temps à le préciser à chaque fois.

Il suffit alors de définir la fonction en précisant la valeur de chaque argument :

```
def calcul_acceleration(dt=0.04, vx1=0, vx2=0, vy1=0, vy2=0, vz1=0, vz2=0):
```

et l'appel de la fonction pour les vecteurs $\vec{v_1} = (0, 1, 0)$ et $\vec{v_2} = (1, 0, 0)$ pour l'intervalle « classique » de nos tables à coussin d'air $\Delta t = 0.04$ se fait de la manière suivante :

```
calcul_acceleration(v1y=1, v2x=1)
```

[7]:
```
# Exemple complet
import math
def calcul_acceleration(dt=0.04, vx1=0, vx2=0, vy1=0, vy2=0, vz1=0, vz2=0):
    """
    Calcul de l'accélération moyenne à partir des coordonnées de
    deux vecteurs vitesse v1 et v2 et d'un intervalle de temps dt

    :param dt: intervalle de temps en secondes (défaut : 0,04 s)
    :param vx1: vecteur v1, coordonnée x (en mètres, défaut : 0 m)
    :param vx2: vecteur v2, coordonnée x (en mètres, défaut : 0 m)
    :param vy1: vecteur v1, coordonnée y (en mètres, défaut : 0 m)
    :param vy2: vecteur v2, coordonnée y (en mètres, défaut : 0 m)
    :param vz1: vecteur v1, coordonnée z (en mètres, défaut : 0 m)
    :param vz2: vecteur v2, coordonnée z (en mètres, défaut : 0 m)
    :return:
    """
    return math.sqrt(((vx2-vx1)/dt)**2 + ((vy2-vy1)/dt)**2 + ((vz2-vz1)/dt)**2)

# Différentes manières d'appeler cette fonction
print("Calcul de l'accélération pour dt=1s, v1=(1,1,0) et v2=(2,0,0)")
# pas très clair, cauchemard de mémoire
print("Accélération : ", calcul_acceleration(1, 1, 2, 1, 0, 0, 0)      , "m/s^2")
# peut être encore moins clair ?...
print("Accélération : ", calcul_acceleration(1, 1, 2, 1)               , "m/s^2")
# mieux, non ?
print("Accélération : ", calcul_acceleration(vx1=1, vy1=1, vx2=2, dt=1), "m/s^2")
# on peut mixer les arguments par position et par nom
```

(continues on next page)

(suite de la page précédente)

```
print("Accélération : ", calcul_acceleration(1, vx1=1, vy1=1, vx2=2)    , "m/s^2")
```

```
Calcul de l'accélération pour dt=1s, v1=(1,1,0) et v2=(2,0,0)
Accélération :  1.4142135623730951 m/s^2
Accélération :  1.4142135623730951 m/s^2
Accélération :  1.4142135623730951 m/s^2
Accélération :  1.4142135623730951 m/s^2
```

#### 2.1.3.3 Variables locales et globales (ou portée des variables)

Note : La portée d'une variable correspond aux parties du programme où la variable est définie.

##### 2.1.3.3.1 Définitions

- Une variable définie à l'intérieur d'une fonction est une variable **locale**. Elle ne sera pas reconnue au sein d'une autre fonction ni au sein du programme principal.
- Une variable définie au niveau du programme principal est une variable **globale**. Elle est reconnue partout dans le programme, même au sein des fonctions.

Le programme ci-dessous illustre ces deux premiers points :

- la variable globale q est bien reconnue par la fonction qui peut ainsi l'afficher.
- La variable locale p (définie au sein de la fonction) n'est pas reconnue dans le programme principal qui ne peut pas exécuter la demande d'affichage..

```
1 def fonct1():
2     p = 20              # ici p est une variable locale
3     print("affichage 1 : ", p,q)    # la variable globale q est bien reconnue par fonct1
4
5 q=10                          # q est une variable globale
6 fonct1()
7 print("affichage 2 : ", p,q)
```

```
affichage 1 :  20 10

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-78fd6979af90> in <module>
      5 q=10                          # q est une variable globale
      6 fonct1()
----> 7 print("affichage 2 : ", p,q)

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

### Priorités

Lorsque des variables locales et globales sont définies par un même nom au sein d'un programme, elles obéissent à des règles de priorité :

- au sein d'une fonction, ce sont les variables définies localement qui ont la priorité sur les variables globales. Ainsi, lors de l'exécution d'une fonction, c'est la valeur de la variable locale qui est prise en compte.
- au sein du programme principal, une variable globale conserve sa valeur initiale même si elle a été modifiée au sein d'une fonction.

[8]:
```
def fonct1():
    p = 20              # ici p est une variable locale qui prend la priorité
    print("affichage 2 : p = ", p)

p=15                  # ici p est une variable globale dont la valeur est 15
print("affichage 1 : p = ", p)
fonct1()
print("affichage 3 : p = ", p)     # Au sein du programme principal, la variable␣
↪globale p garde sa valeur initiale
```

(continues on next page)

(suite de la page précédente)

```
affichage 1 : p =  15
affichage 2 : p =  20
affichage 3 : p =  15
```

#### 2.1.3.3.2 Avantages des variables locales

Les variables locales permettent ainsi de compartimenter les actions. Cela signifie qu'un programme peut contenir quantités de fonctions sans se préoccuper le moins du monde des noms de variables qui y sont utilisées : en effet, ces variables étant locales (définies uniquement au sein d'une fonction), elles ne peuvent jamais interférer avec d'autres variables locales définies dans d'autres fonctions.

[9]:
```
def fonct1():
    p = 20              # ici p est une variable locale
    print("affichage 1 : p = ", p)

def fonct2():
    p = 10              # ici p est une variable locale différente de celle créée␣
↪dans fonct1
    print("affichage 2 : p = ", p)

fonct1()            # appel et exécution de la fonction fonct1
fonct2()            # appel et exécution de la fonction fonct2
```

```
affichage 1 : p =  20
affichage 2 : p =  10
```

#### 2.1.3.3.3 Conversion d'une variable locale en variable globale - inconvénients

On peut faire en sorte qu'une variable déclarée au sein d'une fonction soit malgré tout une variable globale. Pour cela, il suffit d'utiliser l'instruction **global**.

Remarque : il est cependant préférable d'éviter l'utilisation de l'instruction **global** car c'est une source d'erreurs (on peut ainsi modifier le contenu d'une variable globale en croyant agir sur une variable locale).

La sagesse recommande donc de suivre la règle suivante : ne jamais affecter dans une fonction une variable de même nom qu'une variable globale.

[10]:
```
def fonct1():
    global p          # ici p est une variable globale
    p=20              # Notez l'absence de l'instruction return pour la variable␣
↪globale!

def fonct2():
    q=20              # ici q est une variable locale
                      # Notez l'absence de l'instruction return et sa conséquence␣
↪en ligne 16-17!

def fonct3():
    r=20              # ici r est une variable locale
    return(r)         # L'instruction return permet ici de renvoyer le contenu de␣
↪la variable r au programmme principal

fonct1()
print("affichage 1 : p = ", p)  # Au sein du programme principal, la variable␣
↪globale p est bien reconnue alors qu'elle
                                # n'était définie qu'au sein de la fonction fonct1.
res2=fonct2()
```

(continues on next page)

(suite de la page précédente)

```
print("affichage 2 : q = ", res2)  # La fonction fonct2 ne renvoie rien et la␣
↪variable q est locale à la fonct2 :
                                   # La variable res2 ne contient donc aucune␣
↪valeur !

res3=fonct3()                      # La fonction fonct3 renvoie le contenu de la␣
↪variable r locale
print("affichage 2 : r = ", res3)  # La variable res3  contient donc la valeur 20!
```

```
affichage 1 : p =  20
affichage 2 : q =  None
affichage 2 : r =  20
```

### 2.1.4 Quelques opérations basiques en Python

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

#### 2.1.4.1 Opérations mathématiques usuelles

quotient de la division euclidienne : `//`

[1]:
```
25//7
```

[1]:
```
3
```

Reste de la division euclidienne `%`

[2]:
```
25%7
```

[2]:
```
4
```

puissance `**`

[3]:
```
3**3
```

[3]:
```
27
```

racine carrée : `sqrt` (importer au préalable la fonction)

[4]:
```
# Plusieurs méthodes (au choix) pour importer

# On importe l'ensemble de la bibliothèque math dans le programme
from math import *
print(sqrt(3))

# On importe juste la fonction nécessaire
from math import sqrt
print(sqrt(3))

# On importe l'intégralité de la bibliothèque
# mais on doit préfixer la fonction par le nom de la bibliothèque
import math
print(math.sqrt(3))
```

```
1.7320508075688772
1.7320508075688772
1.7320508075688772
```

#### 2.1.4.2 Autres opérations mathématiques

A partir de la bibliothèque math :
- sin
- cos
- log (logarithme népérien)
- log10 (logarithme décimal)
- exp
- abs (pour valeur absolue)
- min
- max
- floor (partie entière)

A partir de la bibliothèque random :
- random : génère un nombre aléatoire décimal entre 0 et 1

[5]:
```
from math import *
sqrt(3)
```

[5]:
```
1.7320508075688772
```

[6]:
```
from math import*
sin(30)
# attention les angles sont en radian
```

[6]:
```
-0.9880316240928618
```

[7]:
```
from math import*
log(10)
abs(-2)
```

[7]:
```
2
```

#### 2.1.4.3 Tests et opérateurs logiques

Les tests suivants sont disponibles
- == : vérifie une égalité
- != : différent
- < : inférieur
- > : supérieur
- >= : supérieur ou égal
- <= : inférieur ou égal

Les opérateurs logiques ou opérateurs booléens correspondent aux opérations logiques suivantes :
- `and` : et
- `or` : ou
- `not` : non

La réponse possible à ces différentes opérations est : vraie (1) ou fausse (0). L'exemple ci-dessous utilise l'opération « ou » pour déterminer si l'on peut calculer la somme de deux fractions dans un premier temps. Si la réponse est positive alors les valeurs du numérateur et du dénominateur sont affichées.

[8]:
```
def somme_fractions(a,b,c,d):
    print("Somme de {0}/{1} + {2}/{3}".format(a,b,c,d))
    if b==0 or d==0:
        print("  -> Impossible : dénominateur nul interdit")
```

(continues on next page)

(suite de la page précédente)

```
    else:
        print("   -> Résultat : {0}/{1}".format(a*d+c*d, b*d))

somme_fractions(1,3,1,0)
somme_fractions(1,3,3,5)
```

```
Somme de 1/3 + 1/0
  -> Impossible : dénominateur nul interdit
Somme de 1/3 + 3/5
   -> Résultat : 20/15
```

### 2.1.5 Les tests en python

(code sous licence creative commun CC BY-NC-SA BY Alexis Dendiével)

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Les tests constituent, avec les boucles, l'une des bases fondamentales de la programmation. Régulièrement en effet, nous devons faire des tests afin de dérouler l'algorithme.

— Si telle condition est vrai alors faire ceci
— sinon faire cela

#### 2.1.5.1 if / else

[1]:
```
n = 2
if n > 3:
    print("n est supérieur à 3")
else:
    print("n est inférieur ou égal à 3")
```

```
n est inférieur ou égal à 3
```

Un exemple lié à la classification périodique des éléments : On se propose de savoir si l'élément, défini par son numéro atomique, apparient ou non aux trois premières linges de la classification

[2]:
```
# premier test

def ligne(Z):
    """
    Dit si un élément fait partie des trois premières lignes du tableau périodique.

    :param Z: numéro atomique
    """
    # test vérifiant si l'élément fait partie des trois premières lignes
    print("Elément de numéro atomique", Z, ":")
    if Z<= 18:
        print("  fait partie des trois premières lignes de la classification")
    else:
        print("  ne fait pas partie des trois premières lignes de la classification
↪")

ligne(3)
ligne(5)
```

```
Elément de numéro atomique 3 :
  fait partie des trois premières lignes de la classification
```

(continues on next page)

(suite de la page précédente)

```
Elément de numéro atomique 5 :
  fait partie des trois premières lignes de la classification
```

La syntaxe traduit l'algorythme en langage python :

prenons un exemple plus conséquent afin de voir l'utilité du test if en contexte. Il s'agit d'un programme calculant les forces électromagnétiques et gravitationnelles, et les comparant. A la lecture du programme, vous trouverez deux tests : - le premier donne le côté attractif ou répulsif de la force électromagnétique - le second compare les deux forces afin de définir la prédominance

[3]:
```
# comparaison des forces de gravitation et électromagnétique
# entre deux masses m1 et m2 de charge q1 et q2 séparées d'une distance d

def comparaison_forces(m1, e1, m2, e2, d):
    """
    Comparaison des forces de gravitation et électromagnétique entre deux
    masses m1 et m2 (en kg), de charge q1 et q2 (en multiple de e, la
    charge élémentaire), séparées d'une distance d (en mètres)

    :param m1: masse m1 (en kg)
    :param e1: charge q1 (en multiple de e)
    :param m2: masse m2 (en kg)
    :param e2: charge q2 (en multiple de e)
    :param d: distance séparant m1 et m2 (en mètres)
        """
    # les constantes utilisées
    G = 6.67e-11
    k = 9.0e9
    e = 1.6e-19

    # présentation du programme
    print("Comparaison des forces de gravitation et électromagnétiques : ")
    print("  - objet 1 : masse {0:.2e} kg, charge {1:.2e} C".format(m1, e1*e))
    print("  - objet 2 : masse {0:.2e} kg, charge {1:.2e} C".format(m2, e2*e))

    # caractère attractif ou répulsif
    if e1*e2 > 0:
        caractere = "répulsive"
    else:
        caractere = "attractif"

    # calcul des forces
    FG = G*m1*m2/d**2
    FE = abs(k*e*e1*e*e2/d**2)

    # comparaison des forces
    if FG > FE:
        preponderant = "force de gravitation"
        comparaison = FG/FE
    else:
        preponderant = "force électromagnétique"
        comparaison = FE/FG

    # impression des résultats

    print("\nRésultat:\n")
    print("- la valeur de la force de gravitation est: {0:.2e}".format(FG), ' N')
    print("    cette force est attractive")
    print("- la valeur de la force électromagnétique est: {0:.2e}".format(FE), ' N
↪')
    print("    cette force est", caractere)
```

(continues on next page)

(suite de la page précédente)

```
    print("- la", preponderant, "est prépondérante")
    print("- le rapport de ces deux forces est: {0:.2e}".format(comparaison))


# Utilisation de la fonction
comparaison_forces(m1=5e-7, e1=400, m2=5e-7, e2=400, d=3e-6)
```

```
Comparaison des forces de gravitation et électromagnétiques :
  - objet 1 : masse 5.00e-07 kg, charge 6.40e-17 C
  - objet 2 : masse 5.00e-07 kg, charge 6.40e-17 C

Résultat:

- la valeur de la force de gravitation est: 1.85e-12  N
    cette force est attractive
- la valeur de la force électromagnétique est: 4.10e-12  N
    cette force est répulsive
- la force électromagnétique est prépondérante
- le rapport de ces deux forces est: 2.21e+00
```

Plusieurs points : - Dans ces exemples, le else n'est en rien obligatoire.

#### 2.1.5.2 if / elif / else

On souhaite parfois faire plusieurs tests imbriqués, on utilise pour cela l'instruction elif (else if)

[4]:
```
# test imbriqué
def ligne(Z):
    """
    Indique la ligne dont fait partie l'élément

    :param Z: numéro atomique
    """
    print("Elément de numéro atomique Z = ",Z)
    # test vérifiant si l'élément fait partie des trois premières lignes
    if Z<= 2:
        print("  fait partie de la première ligne de la classification")
    elif Z <=10:
        print("  fait partie de la deuxième ligne de la classification")
    elif Z <=18:
        print("  fait partie de la troisième ligne de la classification")
    else:
        print("   ne fait pas partie des trois premières lignes de la␣
↪classification")

# Utilisation
ligne(7)
ligne(1)
ligne(16)
ligne(33)
```

```
Elément de numéro atomique Z =  7
  fait partie de la deuxième ligne de la classification
Elément de numéro atomique Z =  1
  fait partie de la première ligne de la classification
Elément de numéro atomique Z =  16
  fait partie de la troisième ligne de la classification
Elément de numéro atomique Z =  33
   ne fait pas partie des trois premières lignes de la classification
```

Attention, erreur fréquente : le test égal se note ==, et pas = (assignation de variable)

### 2.1.6 Les boucles en python

(code sous licence creative commun CC BY-NC-SA BY Alexis Dendiével)

Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

Les bouclent constituent, avec les tests, un élément indispensable de programmation. Il existe deux types de boucles : - La boucle for qui se réitère quand on parcourt un itérable par exemple, - la boucle while qui se réitère tant qu'une condition est remplie

voyons cela par l'exemple

#### 2.1.6.1 La boucle for

[1]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

Dans cet exemple, nous voyons que i prend successivement les valeurs allant de 0 à 9 L'utilité peut être de faire plusieurs fois le même calcul, comme par exemple le calcul de la distance de chute.

[2]:
```
# donnée
g = 9.81

# calcul de l'ordonnée d'un laché vertical sans vitesse initial dans le champ de␣
↪pesanteur
for t in range (10):
    y = 0.5*g*t**2
    print("à l'instant ", t, ", y vaut : ", y, " m")
```

```
à l'instant  0 , y vaut :  0.0  m
à l'instant  1 , y vaut :  4.905  m
à l'instant  2 , y vaut :  19.62  m
à l'instant  3 , y vaut :  44.145  m
à l'instant  4 , y vaut :  78.48  m
à l'instant  5 , y vaut :  122.625  m
à l'instant  6 , y vaut :  176.58  m
à l'instant  7 , y vaut :  240.345  m
à l'instant  8 , y vaut :  313.92  m
à l'instant  9 , y vaut :  397.305  m
```

la boucle for peut aussi servir à parcourir un itérable, comme le montre l'exemple suivant, qui donne les n premiers éléments chimiques avec leur symbole.

[3]:
```
# liste des 18 premiers éléments chimiques de la classification
elements = [ "Hydrogène H", "Hélium He", "Lithium Li", "Béryllium Be",
"Bore B", "Carbone C", "Azote N", "Oxygène O", "Fluor F", "Néon Ne",
"Sodium Na", "Magnésium Mg", "Aluminium Al", "Silicium Si", "Phosphore P",
"Soufre S", "Chlore Cl", "Argon Ar"]
```

(continues on next page)

(suite de la page précédente)

```
# la boucle qui parcours la liste des éléments et les imprime
for element in elements:
    print (element)
```

```
Hydrogène H
Hélium He
Lithium Li
Béryllium Be
Bore B
Carbone C
Azote N
Oxygène O
Fluor F
Néon Ne
Sodium Na
Magnésium Mg
Aluminium Al
Silicium Si
Phosphore P
Soufre S
Chlore Cl
Argon Ar
```

On peut bien sur combiner cette boucle avec un test permettant de n'imprimer que les éléments de numéro atomique inférieur à une certaine valeur :

[4]:
```
def affichage_premiers_elements(Zmax):
    """
    Affiche les éléments de numéro atomique inférieur ou égal à Z_max

    :param Z_max: numéro atomique maximal (inférieur à 18)
    """
    # liste des 18 premiers éléments chimiques de la classification
    elements = [
    "Hydrogène H", "Hélium He", "Lithium Li", "Béryllium Be", "Bore B",
    "Carbone C", "Azote N", "Oxygène O", "Fluor F", "Néon Ne", "Sodium Na",
    "Magnésium Mg", "Aluminium Al", "Silicium Si", "Phosphore P", "Soufre S",
    "Chlore Cl", "Argon Ar"]

    # test pour vérifier que Z est inférieur ou égal à 18
    if Zmax <=18:
        print("les ", Zmax, "premiers éléments de la classification sont :")
        for i in range(Zmax):
            print (" - ",elements[i])
    else:
        print("tsss...lisez la doc : le numéro atomique doit être inférieur à 18 !!
↪")

affichage_premiers_elements(4)
print("")
affichage_premiers_elements(33)
```

```
les  4 premiers éléments de la classification sont :
 -  Hydrogène H
 -  Hélium He
 -  Lithium Li
 -  Béryllium Be

tsss...lisez la doc : le numéro atomique doit être inférieur à 18 !!
```

#### 2.1.6.2 La boucle while

C'est une autre manière de programmer une boucle : celle-ci aura lieu tant qu'une condition sera remplie. Reprenons l'exemple précédant avec une boucle while :

[5]:
```
def affichage_premiers_elements_avec_while(Zmax):
    """
    Affiche les éléments de numéro atomique inférieur ou égal à Z_max

    :param Z_max: numéro atomique maximal (inférieur à 18)
    """
    # liste des 18 premiers éléments chimiques de la classification
    elements = [ "Hydrogène H", "Hélium He", "Lithium Li", "Béryllium Be",
    "Bore B", "Carbone C", "Azote N", "Oxygène O", "Fluor F", "Néon Ne",
    "Sodium Na", "Magnésium Mg", "Aluminium Al", "Silicium Si", "Phosphore P",
    "Soufre S", "Chlore Cl", "Argon Ar"]

    # test pour vérifier que Z est inférieur ou égal à 18
    if Zmax <=18:
        print("les ", Zmax, "premiers éléments de la classification sont :")
        numero = 1
        while numero <=Zmax:
            print (" - ",elements[numero-1])
            numero = numero + 1
    else:
        print("nan, j'affiche toujours pas : le numéro atomique est supérieur à 18
↪")

affichage_premiers_elements_avec_while(4)
print("")
affichage_premiers_elements_avec_while(32)
```

```
les  4 premiers éléments de la classification sont :
 -  Hydrogène H
 -  Hélium He
 -  Lithium Li
 -  Béryllium Be

nan, j'affiche toujours pas : le numéro atomique est supérieur à 18
```

#### 2.1.6.3 Complément

Il peut être utile de vouloir sauter une étape à l'intérieur d'une boucle avant qu'elle ne s'achève. On utilise pour cela : - l'instruction continue

[6]:
```
for i in range(10):
    if i ==5:
        continue
    print(i, " ")
```

```
0
1
2
3
4
6
7
8
9
```

Nous voyons ici que l'impression du 5 n'a pas été effective. Nous pouvons aussi vouloir sortir d'une boucle, pour cela : - on utilise l'instruction break

[7]:
```
for i in range(10):
    if i ==5:
        break
    print(i, " ")
```

```
0
1
2
3
4
```

Ces deux instructions, continue et break, peuvent également s'utiliser avec la boucle while.

### 2.1.7 Les listes



`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Comme son nom l'indique, une liste permet de lister différents éléments.

Les éléments d'une liste :
- s'écrivent entre crochets,
- sont séparés par une virgule,
- peuvent être de nature différente (chaine de caractères, nombre entier, nombre réel,...)
- sont repérés par leur position dans la chaîne, appelée indice.

**ATTENTION : le premier indice d'une liste a pour valeur 0 !**

Remarque sur l'organisation de ce fichier noteboook : très souvent, les cellules fonctionnent par paire ; elles portent alors le même titre. La première cellule explique le cours et la suivante est une cellule de codes illustrant le cours.

Remarque supplémentaire :

Il existe des listes non modifiables appelées n-uplets (ou tuple en anglais).

Toutes les méthodes ou fonctions décrites ci-dessous et permettant de modifier les éléments d'une liste ne sont bien sûr pas applicables à un n-uplet, puisque, par définition, un n-uplet n'est pas modifiable.

#### 2.1.7.1 Créer des listes

*L = [5,2,8,17,6,14]*

Crée une liste nommée L contenant les éléments 5, 2, 8, 17, 6 et 14.

L'initialisation d'un n-uplet s'effectue différemment. Il faut écrire les éléments entre parenthèses et non pas entre crochets comme pour les listes.

*upl=(1,"a",3)*

Le n-uplet nommé upl contient les éléments 1, a et 3.

[1]:
```
# Plusieurs manières de créer des listes

L = [5,2,8,17,6,14]
print(L)
upl=(1,'a',3)
print(upl)
```

```
[5, 2, 8, 17, 6, 14]
(1, 'a', 3)
```

#### 2.1.7.2 Récupérer l'indice d'un élément d'une liste

La méthode L.index() permet de récupérer l'indice d'un élément d'une liste. Attention, dans le cas où plusieurs éléments ont la même valeur, cette méthode renvoie l'indice de l'élément d'indice le plus petit.

[2]:
```
L = [5,2,8,17,8,14]
a=L.index(5)
print(a)
b=L.index(8)
print(b)
```

```
0
2
```

#### 2.1.7.3 Récupérer les éléments d'une liste

ATTENTION : l'accès à un élément d'un n-uplet par indice s'effectue grâce à des crochets comme pour les listes.

Nous allons travailler sur l'exemple de la liste créée ci-dessus :

```
L[0]            renvoie le premier élément, ici 5
L[2]            renvoie l'élément d'indice 2 (en 3ème position donc), ici 8
L[-1]           renvoie le dernier élément, ici 14
L[-2]           renvoie l'avant-dernier élément, ici 6
L[1:3]          renvoie les éléments d'indice 1 et 2  (ATTENTION : indice 3 non␣
↪inclus)
L[3:]           renvoie les éléments dont les indices sont supérieurs à 3.
L[:4]           renvoie les éléments dont les indices ont pour valeur : 0, 1, 2 et␣
↪3.
len(L)          renvoie la longueur de la liste, ici 6
L1 = []         crée une liste vide
```

[3]:
```
# Plusieurs manières de récupérer les éléments d'une liste
# Applications

L=[5,2,8,17,6,14]            # création d'une liste
upl=(1,'a',3)                #  création d'un n-uplet
print(L)
print(L[0])
print(L[2])
print(L[-1])
print(L[-2])
print(L[1:3])
print(L[3:])
print(L[:4])
print(len(L))
print(len(upl))
print(upl[1])
L1=[]
print(L1)
```

```
[5, 2, 8, 17, 6, 14]
5
8
14
6
[2, 8]
```

(continues on next page)

(suite de la page précédente)

```
[17, 6, 14]
[5, 2, 8, 17]
6
3
a
[]
```

#### 2.1.7.4 Ajouter une valeur ou supprimer une valeur d'une liste

*Les méthodes modifient les listes et leur syntaxe est toujours similaire : L.méthode()*

[4]:
```
# PLusieurs méthodes pour ajouter une valeur ou supprimer des valeurs d'une liste
#(cours et applications)

L=[5,2,8,17,6,14]
print(L)
L.append(20)                # Ajoute l'élément 20 à la fin de la liste L.
print(L)
L.insert(2,20)              # Insère l'élément 20 à la position d'indice 2
print(L)                    #... (donc en troisième position) de la liste L.
L.remove(20)                # Supprime la première occurrence (apparition)
print(L)                    #...de l'élément 20 dans la liste L.
L.pop(-1)                   # Supprime l'élément d'indice -1
print(L)                    #... (donc le dernier élément) de la liste L.
```

```
[5, 2, 8, 17, 6, 14]
[5, 2, 8, 17, 6, 14, 20]
[5, 2, 20, 8, 17, 6, 14, 20]
[5, 2, 8, 17, 6, 14, 20]
[5, 2, 8, 17, 6, 14]
```

#### 2.1.7.5 Analyser le contenu d'une liste

*Les fonctions ne modifient pas les listes et leur syntaxe est toujours similaire : fonction(L)*

*Les méthodes modifient les listes et leur syntaxe est toujours similaire : L.méthode()*

- min(L) : Renvoie le plus petit élément de la liste L.
- max(L) : Renvoie le plus grand élément de la liste L.
- sorted(L) : Renvoie une copie triée de la liste contenant les éléments de la liste L rangés par ordre croissant ou alphabétique. MAIS, la liste L n'est pas modifiée !
- sorted(L,reverse=True) : Renvoie une copie triée de la liste contenant les éléments de la liste L rangés par ordre décroissant ou inverse du sens alphabétique. MAIS, la liste L n'est pas modifiée !
- L.sort() : Modifie la liste L qui dorénavant contiendra les éléments triés (mais ne la renvoie pas).

Remarque : sort() est une méthode et non une fonction... D'où sa syntaxe différente.

- choice(L) : Renvoie en choisissant au hasard un élément de la liste L. Cette fonction appartient au module random.
- sample(L,3) : Retourne une liste de 3 éléments choisis aléatoirement et sans remise dans la liste L. Cette fonction appartient au module random.

[5]:
```
# Quelques fonctions permettant d'analyser le contenu d'une liste
# Applications

from random import choice, sample
# Les fonctions sample et choice appartiennent à la bibliothèque random

L=[5,2,8,17,6,14]
print("L = ",L)
print(min(L))
```

(continues on next page)

(suite de la page précédente)

```
print(max(L))
Ltrie_endroit=sorted(L)
print("la liste Ltrie_endroit est une copie triée de L :  ",Ltrie_endroit)
print("Comme vous pouvez le constater, la liste L n'est pas modifiée : L = : ",L)
Ltrie_envers=sorted(L,reverse=True)
print("la liste Ltrie_envers est une copie triée de L :  ",Ltrie_envers)
print("Comme vous pouvez le constater, la liste L n'est pas modifiée : L =",L)
L.sort()
print("Comme vous pouvez le constater, la liste L est modifiée : L =",L)
print(choice(L))
print(sample(L,3))
```

```
L =  [5, 2, 8, 17, 6, 14]
2
17
la liste Ltrie_endroit est une copie triée de L :   [2, 5, 6, 8, 14, 17]
Comme vous pouvez le constater, la liste L n'est pas modifiée : L = :  [5, 2, 8,␣
↪17, 6, 14]
la liste Ltrie_envers est une copie triée de L :   [17, 14, 8, 6, 5, 2]
Comme vous pouvez le constater, la liste L n'est pas modifiée : L = [5, 2, 8, 17,␣
↪6, 14]
Comme vous pouvez le constater, la liste L est modifiée : L = [2, 5, 6, 8, 14, 17]
6
[5, 2, 17]
```

#### 2.1.7.6 Parcourir le contenu d'une liste

La boucle « for » est particulièrement bien adaptée aux listes de valeurs.

Soit L une liste de longueur n :

→ Si on a besoin de parcourir une liste élément par élément grâce à leur indice pour ensuite utiliser une instruction faisant intervenir cet indice, on utilise l'instruction : **for i in range(0, len(L)) :**

la variable i (qui représente l'indice d'un élément) prendra les valeurs de 0 à len(L)-1 soit de 0 à n-1

→ Si on a besoin de parcourir une liste, élément par élément, en nous intéressant uniquement à leur valeur pour ensuite utiliser une instruction permettant de travailler sur ces valeurs, on peut utiliser l'instruction : **for x in L :**

la variable x prendra l'une après l'autre toutes les valeurs de la liste L.

Remarque 1 : On peut toujours utiliser la première instruction à la place de la deuxième, mais pas l'inverse !

Remarque 2 : les lettres i, j et k sont traditionnellement utilisées pour désigner les indices alors que les autres lettres désignent des variables. Par exemple, ici, la lettre x parcourt les valeurs de L.

[6]:
```
#Comment parcourir le contenu d'une liste : applications de base

L=[5,2,8,17,6,14]
print("premier exemple :")
for i in range(0,len(L)):
# i sera égal à tous les indices de la liste (ici de 0 à 5)
    if i%2==0 :  print(L[i])
# test permettant de sélectionner les indices i pairs
#...(reste de la division de i par 2 vaut 0)

print("second exemple :")
for x in L :
# x balaye toutes les valeurs de la liste (ici 5, puis 2,...)
    if x > 7:
        print(x)
```

```
premier exemple :
5
8
6
second exemple :
8
17
14
```

[7]:
```
#Comment parcourir le contenu d'une liste : applications plus poussées

print("premier exemple :")
L=[5,2,8,17,6,14]
for i in range (0,len(L)):
    L[i] = L[i] + 1   # on additionne 1 à chaque valeur de la liste
print(L)

print("second exemple :")
L=[5,2,8,17,6,14]
# création d'une nouvelle variable s qui est initialisée à 0
s = 0
for x in L :
    s = s + x
# après le premier passage dans la boucle, s sera égal à
# ...son ancienne valeur (0) additionné à x
    print (" s intermédiaire : ", s)
print (" Somme finale : ", s)

# test pour savoir si la valeur 8 est dans la liste
if 8 in L : print("le nombre 8 est présent dans la liste")
else : print("le nombre 8 n'est pas présent dans la liste")
```

```
premier exemple :
[6, 3, 9, 18, 7, 15]
second exemple :
 s intermédiaire :  5
 s intermédiaire :  7
 s intermédiaire :  15
 s intermédiaire :  32
 s intermédiaire :  38
 s intermédiaire :  52
 Somme finale :  52
le nombre 8 est présent dans la liste
```

#### 2.1.7.7 Concaténation de listes

On peut concaténer (mettre bout à bout) des listes, et ce, de plusieurs manières différentes.

En voici quelques exemples : (L1, L2 et L sont des listes)

L = 3 * L1 : L sera une liste contenant les éléments de L1, répétés 3 fois. Elle sera donc trois fois plus longue.

L = L1 + L2 : L sera la concaténation de L1 et L2. elle contiendra d'abord les éléments de L1, puis ceux de L2

[8]:
```
# Opérations mathématiques sur les listes : applications
L1=[1,2,3]
L2=[4,5,6]
L=3*L1
print(L1)
print(L)
L=L1+L2
print(L)
```

```
[1, 2, 3]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
[1, 2, 3, 4, 5, 6]
```

#### 2.1.7.8 Conversion d'une chaîne de caractères vers une liste de caractères ou l'inverse

→ D'une chaîne de caractères vers une liste de caractères

```
ch='ISN'  :     Création d'une chaîne de caractères nommée ch
                et assignation de cette chaîne avec les caractères ISN

list(ch)  :     Convertit la chaîne de caractères ch en liste de caractères
```

→ D'une liste de caractères vers une chaîne de caractères

Attention : la liste doit être forcément constituée de caractères

```
L = ['I','S','N']  :   crée une liste contenant 3 éléments de type caractère
'sep'.join(L)  :  renvoie une chaîne de caractères obtenue en concaténant les
                  éléments de la liste L séparés par le séparateur sep.
```

[9]:
```
ch='ISN'
chbis=list(ch)
print(ch)
print(chbis)

L=['I','S','N']
chter='-'.join(L)       # ici le séparateur est un tiret -
print(L)
print(chter)
```

```
ISN
['I', 'S', 'N']
['I', 'S', 'N']
I-S-N
```

#### 2.1.7.9 Listes de listes (tableaux)

Nous allons travailler sur un nouvel exemple

```
tableau = [['Anne','Tom','Léo','Eva'], [6,7,8,9],[10,20,30,40]]  :
crée un tableau contenant 3 listes de 4 éléments chacune

Tableau[0]                      # renvoie la première liste
Tableau[i][j]                   # renvoie le jème élément de la ième liste
```

[10]:
```
# On peut créer des listes de listes (donc un tableau !)

tableau = [['Anne','Tom','Léo','Eva'], [6,7,8,9],[10,20,30,40]]
print(tableau)
print(tableau[0])
print(tableau[0][0])
print(tableau[1][2])
print(tableau[-1][-1])
print(tableau[-1][0])
```

```
[['Anne', 'Tom', 'Léo', 'Eva'], [6, 7, 8, 9], [10, 20, 30, 40]]
['Anne', 'Tom', 'Léo', 'Eva']
Anne
```

(continues on next page)

(suite de la page précédente)

```
8
40
10
```

### 2.1.8 Les tableaux numpy



`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Dans le cadre du programme du lycée et par souci de simplification, nous ne créeront que des tableaux numpy à une dimension (une ligne).

Ces tableaux ressemblent grandement à des listes mais permettent de faire des opérations dites vectorisées bien plus intuitives et d'éviter les boucles. Comme pour une liste, un tableau numpy est un objet mutable qui permet de lister différents éléments, par contre ces derniers seront obligatoirement tous du même type (entiers, décimaux, chaînes de caractères, listes, tuples...) et la longueur du tableau est non-modifiable.

ATTENTION : comme pour une liste, l'indice du premier élément d'un tableau numpy a pour valeur 0 !

Remarque sur l'organisation de ce fichier noteboook : très souvent, les cellules fonctionnent par paire ; elles portent alors le même titre. La première cellule explique le cours et la suivante est une cellule de codes illustrant le cours.

#### 2.1.8.1 Créer des tableaux numpy à une dimension

- *T = np.array([5,2,8,17,6,14])*
  - Convertit une liste contenant les éléments 5, 2, 8, 17, 6, 14 en un tableau numpy contenant les mêmes éléments et dans le même ordre.
- *T1 = numpy.arange(15)*
  - Crée un tableau contenant 15 valeurs entières allant de 0 à 14.
- *T2 = numpy.arange(0.9,8.1,0.5)*
  - Crée un tableau contenant des valeurs séparées de 0.5 comprises dans l'intervalle [0.9 ;8.1].
  - Le premier paramètre précise la valeur initiale du tableau.
  - Les valeurs du tableau seront comprises dans l'intervalle [premier paramètre, second paramètre].
  - Le dernier paramètre indique l'intervalle entre deux valeurs successives du tableau.
- *T3 = numpy.linspace(0,1/4,16)*
  - Crée un tableau contenant 16 valeurs (de 0 à 1/4=0,25).
  - Le premier paramètre précise la valeur initiale du tableau.
  - Le second paramètre précise la valeur finale du tableau.
  - Le dernier paramètre indique le nombre total de valeurs.

[1]:
```
# Plusieurs manières de créer des tableaux numpy à une dimension
import numpy as np

T = np.array([5,2,8,17,6,14])
print(T)
T1 = np.arange(15)
print(T1)
T2 = np.arange(0.9,8.1,0.5)
print(T2)
T3 = np.linspace(0,1/4,16)
print(T3)
```

```
[ 5  2  8 17  6 14]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
[0.9 1.4 1.9 2.4 2.9 3.4 3.9 4.4 4.9 5.4 5.9 6.4 6.9 7.4 7.9]
```

(continues on next page)

(suite de la page précédente)

```
[0.         0.01666667 0.03333333 0.05       0.06666667 0.08333333
 0.1        0.11666667 0.13333333 0.15       0.16666667 0.18333333
 0.2        0.21666667 0.23333333 0.25      ]
```

#### 2.1.8.2 Récupérer les éléments d'un tableau numpy

Nous allons travailler sur un exemple :

- T[0] renvoie le premier élément, ici 5
- T[2] renvoie l'élément d'indice 2 (en 3ème position donc), ici 8
- T[-1] renvoie le dernier élément, ici 14
- T[-2] renvoie l'avant-dernier élément, ici 6
- T[1 :3] renvoie les éléments d'indice 1 et 2 (ATTENTION : indice 3 non inclus)
- T[3 :] renvoie les éléments à partir de l'indice 3.
- len(T) renvoie la longueur du tableau, ici 6
- T4 = np.array([]) crée un tableau vide

[2]:
```
# Plusieurs manières de récupérer les éléments d'un tableau numpy :
# applications

T=np.array([5,2,8,17,6,14])
print(T)
print(T[0])
print(T[2])
print(T[-1])
print(T[-2])
print(T[1:3])
print(T[3:])
print(len(T))
T4 = np.array([])
print(T4)
```

```
[ 5  2  8 17  6 14]
5
8
14
6
[2 8]
[17  6 14]
6
[]
```

#### 2.1.8.3 Ajouter une valeur ou supprimer une valeur d'un tableau numpy

*Les fonctions ne modifient pas les tableaux d'origine mais retournent de nouveaux tableaux en ajoutant ou en supprimant une valeur du tableau d'origine. Leur syntaxe est toujours similaire : np.fonction(T)*

[3]:
```
# PLusieurs fonctions pour ajouter une valeur ou supprimer des valeurs
# d'un tableau numpy (cours et applications)

T=np.array([5,2,8,17,6,14])

print(T)
T1=np.append(T,18)      # Crée un tableau T1 à partir du tableau T
print(T1)               # en ajoutant l'élément 18 à la fin du
print(T)                # tableau T. Ne modifie pas le tableau T.

T=np.append(T,18)       # Ajoute l'élément 18 à la fin du tableau T.
print(T)
```

(continues on next page)

(suite de la page précédente)

```
T=np.insert(T,2,20)     # Insère l'élément 20 à la position d'indice 2
print(T)                # (donc en troisième position) du tableau T.

T=np.delete(T,2)        # Supprime l'élément d'indice 2 (ici l'élément 20)
print(T)                # du tableau T.
T=np.delete(T,-1)       # Supprime l'élément d'indice -1 (donc le dernier
print(T)                # élément) du tableau T.
```

```
[ 5  2  8 17  6 14]
[ 5  2  8 17  6 14 18]
[ 5  2  8 17  6 14]
[ 5  2  8 17  6 14 18]
[ 5  2 20  8 17  6 14 18]
[ 5  2  8 17  6 14 18]
[ 5  2  8 17  6 14]
```

#### 2.1.8.4 Analyser le contenu d'un tableau numpy

*Les fonctions ne modifient pas les tableaux numpy et leur syntaxe est toujours similaire : fonction(T)*

*Les méthodes modifient les tableaux numpy et leur syntaxe est toujours similaire : T.méthode()*

- min(T) ou np.min(T) : Renvoie le plus petit élément du tableau T.
- max(T) ou np.max(T) : Renvoie le plus grand élément du tableau T.
- sorted(T) : Renvoie une copie triée du tableau contenant les éléments du tableau T rangés par ordre croissant ou alphabétique. MAIS, le tableau T n'est pas modifié !
- sorted(T,reverse=True) : Renvoie une copie triée du tableau contenant les éléments du tableau T rangés par ordre décroissant ou inverse du sens alphabétique. MAIS, le tableau T n'est pas modifié !
- T.sort() : Modifie le tableau T qui dorénavant contiendra les éléments triés (mais ne le renvoie pas).

Remarque : sort() est une méthode et non une fonction... D'où sa syntaxe différente.

- choice(T) : Renvoie en choisissant au hasard un élément du tableau T. Cette fonction appartient au module random.

[4]:
```
# Quelques fonctions permettant d'analyser le contenu d'une liste :
# applications

# La fonction choice appartient à la bibliothèque random
from random import choice

T=np.array([5,2,8,17,6,14])
print("T = ",T,'\n')

print(min(T),'et',np.min(T))
print(max(T), 'et',np.max(T),'\n')

Ttrie_endroit=sorted(T)
print("Le tableau Ttrie_endroit est une copie triée de T :\n",
      "Ttrie_endroit = ",Ttrie_endroit)
print("Comme vous pouvez le constater, le tableau T n'est pas modifié :\n",
      "T = : ",T,'\n')

Ttrie_envers=sorted(T,reverse=True)
print("Le tableau Ttrie_envers est une copie triée de T :\n",
      "Ttrie_envers = ",Ttrie_envers)
print("Comme vous pouvez le constater, le tableau T n'est pas modifié :\n",
      "T =",T,'\n')

T.sort()
print("Comme vous pouvez le constater, le tableau T est modifié :\n",
```

(continues on next page)

(suite de la page précédente)

```
      "T =",T,'\n')

print(choice(T))
```

```
T =  [ 5  2  8 17  6 14]

2 et 2
17 et 17

Le tableau Ttrie_endroit est une copie triée de T :
 Ttrie_endroit =  [2, 5, 6, 8, 14, 17]
Comme vous pouvez le constater, le tableau T n'est pas modifié :
 T = :  [ 5  2  8 17  6 14]

Le tableau Ttrie_envers est une copie triée de T :
 Ttrie_envers =  [17, 14, 8, 6, 5, 2]
Comme vous pouvez le constater, le tableau T n'est pas modifié :
 T = [ 5  2  8 17  6 14]

Comme vous pouvez le constater, le tableau T est modifié :
 T = [ 2  5  6  8 14 17]

5
```

#### 2.1.8.5 Parcourir le contenu d'un tableau numpy

Comme pour les listes , la boucle « for » est particulièrement bien adaptée aux tableaux de valeurs.

Soit T un tableau numpy de longueur n :

→ Si on a besoin de parcourir un tableau élément par élément grâce à leur indice pour ensuite utiliser une instruction faisant intervenir cet indice, on utilise l'instruction : **for i in range(0, len(T)) :**

la variable i (qui représente l'indice d'un élément) prendra les valeurs de 0 à len(T)-1 soit de 0 à n-1

→ Si on a besoin de parcourir un tableau, élément par élément, en nous intéressant uniquement à leur valeur pour ensuite utiliser une instruction permettant de travailler sur ces valeurs, on peut utiliser l'instruction : **for x in T :**

la variable x prendra l'une après l'autre toutes les valeurs du tableau T.

Remarque 1 : On peut toujours utiliser la première instruction à la place de la deuxième, mais pas l'inverse !

Remarque 2 : les lettres i, j et k sont traditionnellement utilisées pour désigner les indices alors que les autres lettres désignent des variables. Par exemple, ici, la lettre x parcourt les valeurs de T.

[5]:
```
# Comment parcourir le contenu d'un tableau numpy :
# applications de base

T=np.array([5,2,8,17,6,14])

print("premier exemple :")

# i balaye (sera égal à) tous les indices des cellules du tableau
# un par un (ici de 0 à 5)

for i in range(0,len(T)):

# test permettant de sélectionner les indices i pairs
# (reste de la division de i par 2 vaut 0)

    if i%2==0 :
        print(T[i])
```

(continues on next page)

(suite de la page précédente)

```
print("second exemple :")

# x balaye toutes les valeurs du tableau (ici 5, puis 2, puis 8...)
for x in T :
    if x > 7:
        print(x)
```

```
premier exemple :
5
8
6
second exemple :
8
17
14
```

[6]:
```
# Comment parcourir le contenu d'un tableau :
# applications plus poussées

print("premier exemple :")
T=np.array([5,2,8,17,6,14])
for i in range (0,len(T)):
    T[i] = T[i] + 1          # on additionne 1 à chaque valeur du tableau
print(T)

print("second exemple :")
T=np.array([5,2,8,17,6,14])

# création d'une nouvelle variable s qui est initialisée à 0
s = 0

for x in T :

# après le premier passage dans la boucle, s sera égal à son
# ancienne valeur (0) additionné à x
    s = s + x

    print (" s intermédiaire : ", s)
print (" Somme finale : ", s)

# test pour savoir si la valeur 8 est dans le tableau

if 8 in T : print("le nombre 8 est présent dans le tableau")
else : print("le nombre 8 n'est pas présent dans le tableau")
```

```
premier exemple :
[ 6  3  9 18  7 15]
second exemple :
 s intermédiaire :  5
 s intermédiaire :  7
 s intermédiaire :  15
 s intermédiaire :  32
 s intermédiaire :  38
 s intermédiaire :  52
 Somme finale :  52
le nombre 8 est présent dans le tableau
```

#### 2.1.8.6 Opérations mathématiques sur les tableaux numpy (vectorisation)

Numpy permet de réaliser des opérations directement sur les éléments d'un tableau sans être obligé de créer une boucle for comme pour les listes, d'où la grande utilité d'utiliser des tableaux numpy au lieu de listes dans les programmes liés aux sciences-physiques.

Prenons un exemple : Soit un dipôle ohmique de résistance R = 3 Ω. On veut déterminer les valeurs que prend la tension U pour une intensité I dont les valeurs sont regroupées dans la liste : I = [0, 0.10, 0.20, 0.30]

[7]:
```
# Création de la liste U

# Si l'on fait :
R = 3         # R est forcément un entier sinon une exception
              # (message d'erreur) sera renvoyée lors de
              # l'exécution du code.

I = [0.10, 0.20, 0.30]
U = R*I
print (U)
print ("On obtient la concaténation de 3 fois la liste I \n",
       "donc MAUVAISE METHODE")
```

```
[0.1, 0.2, 0.3, 0.1, 0.2, 0.3, 0.1, 0.2, 0.3]
On obtient la concaténation de 3 fois la liste I
 donc MAUVAISE METHODE
```

[8]:
```
# On doit donc faire :
I = [0.10, 0.20, 0.30]
R = 3
U = []
for i in range (len (I)):
    Ui=R*I[i]
    U.append(Ui)
print (U)
print ("On obtient bien une liste contenant les valeurs de la tension U \n",
       "donc bonne méthode mais demandant 4 lignes de code")
```

```
[0.30000000000000004, 0.6000000000000001, 0.8999999999999999]
On obtient bien une liste contenant les valeurs de la tension U
 donc bonne méthode mais demandant 4 lignes de code
```

[9]:
```
# Regardons maintenant ce que cela donne avec des tableaux numpy.
I=np.array([0.10, 0.20, 0.30])
R=3
U=R*I
print (U)
print ("On obtient un tableau contenant les valeurs de la tension U \n",
       "donc BONNE METHODE ne demandant qu'une seule ligne de code!")
```

```
[0.3 0.6 0.9]
On obtient un tableau contenant les valeurs de la tension U
 donc BONNE METHODE ne demandant qu'une seule ligne de code!
```

De nombreuses opérations mathématiques de base peuvent être réalisées avec des tableaux numpy. De plus, un grand nombre de fonctions de la bibliothèque math sont aussi définies dans la bibliothèque numpy et peuvent être utilisées lors de calculs utilisant les tableaux numpy.

[10]:
```
import math

print (math.pi, math.log(1),math.sqrt(25))

print (np.pi, np.log(1),np.sqrt(25))
```

```
3.141592653589793 0.0 5.0
3.141592653589793 0.0 5.0
```

#### 2.1.8.7 Conversion d'un tableau numpy de caractères vers une chaîne de caractères

Attention : ici, le tableau numpy doit être forcément constitué de caractères

T = np.array(["I","S","N"]) : crée un tableau numpy contenant 3 éléments de type caractère

"sep".join(T) : renvoie une chaîne de caractères obtenue en concaténant les éléments du tableau T séparés par le séparateur sep.

[11]:
```
T=np.array(['I','S','N'])
mot='-'.join(T)       # ici le séparateur est un tiret -
print(T)
print(mot)
```

```
['I' 'S' 'N']
I-S-N
```

#### 2.1.8.8 Création de tableau numpy à deux dimensions

Nous allons travailler sur un nouvel exemple

tableau = np.array ([["Anne","Tom","Léo","Eva"], [6,7,8,9],[10,20,30,40]] :
crée un tableau contenant 3 lignes de 4 éléments chacune donc un tableau de 3 lignes x 4 colonnes

- Tableau[0] : renvoie la première ligne du tableau
- Tableau[i][j] : renvoie le jème élément de la ième ligne du tableau

[12]:
```
# On peut créer des tableaux à plusieurs dimensions

tableau = np.array([['Anne','Tom','Léo','Eva'], [6,7,8,9],[10,20,30,40]])
print("Tableau\n", tableau,'\n')
print("Ligne\n", tableau[0],'\n')
print("Elément d'une ligne\n", tableau[0][0],'\n')
print("Elément d'une ligne\n", tableau[1][2],'\n')
print("Elément d'une ligne, à partir de la fin\n", tableau[-1][-1],'\n')
print("Elément d'une ligne, à partir de la fin\n",tableau[-1][0], "\n")
print("Transposition\n", tableau.T, "\n")
print("Partie de tableau\n", tableau[0:3,1:3], "\n")
print("Colonne\n", tableau[:,0], "\n")
```

```
Tableau
 [['Anne' 'Tom' 'Léo' 'Eva']
 ['6' '7' '8' '9']
 ['10' '20' '30' '40']]

Ligne
 ['Anne' 'Tom' 'Léo' 'Eva']

Elément d'une ligne
 Anne

Elément d'une ligne
 8

Elément d'une ligne, à partir de la fin
```

(continues on next page)

(suite de la page précédente)

```
 40

Elément d'une ligne, à partir de la fin
 10

Transposition
 [['Anne' '6' '10']
 ['Tom' '7' '20']
 ['Léo' '8' '30']
 ['Eva' '9' '40']]

Partie de tableau
 [['Tom' 'Léo']
 ['7' '8']
 ['20' '30']]

Colonne
 ['Anne' '6' '10']
```

### 2.1.9 Import de données numériques



`Télécharger le pdf`

`Télécharger le notebook` et le `fichier csv`

`Lancer le notebook sur binder (lent)`

Le programme présenté ci-dessous est adapté à des fichiers .csv (type tableur) obtenus lors de pointages vidéo. Il devra évidemment être adapté pour des fichiers obtenus lors d'autres expériences.

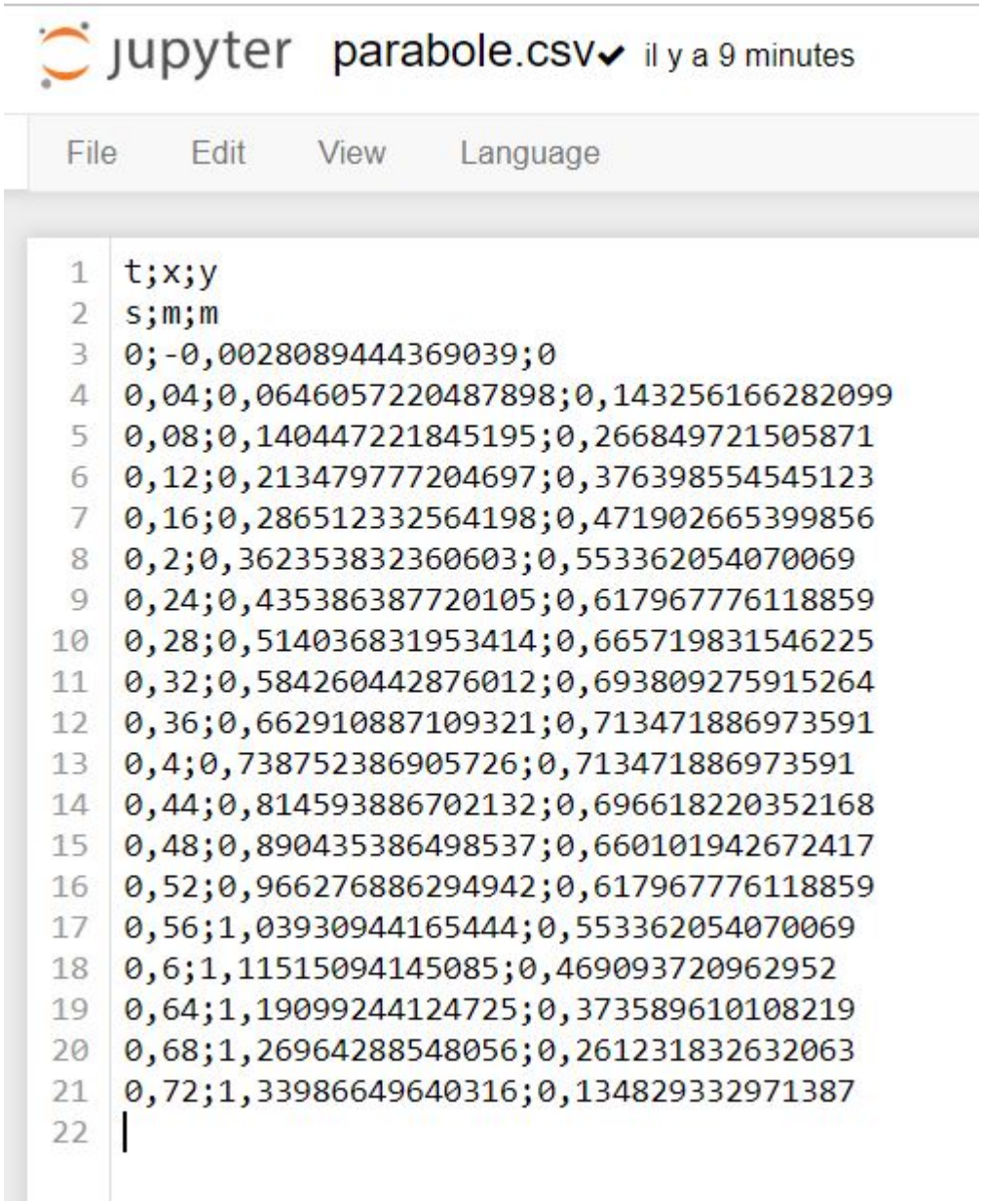
1. Enregistrer ou exporter le fichier contenant votre tableau de données sous format .csv (ou .txt pour Avistep) dans le dossier contenant votre notebook (fichier .ipynb) ou votre programme python (fichier.py). Attention, pour l'utilisation avec l'ENT Nero version 2018, petite subtilité à la fin.
   — Dans Regressi, enregistrer le fichier sous le format (type) OpenOffice, CSV (choisir « Vrai CSV » dans la fenêtre qui s'affiche alors).
   — Dans Loggerpro, exporter le fichier comme CSV…
   — Dans Aviméca, exporter les données dans Regressi puis vous reporter à la ligne ci-dessus.
   — Dans Avistep, exporter/enregistrer le fichier sous le format .txt
   — Dans Excel, enregistrer votre fichier sour le format CSV (séparateur :point-virgule).
   — Dans OpenCalc, enregistrer votre fichier sour le format CSV (texte CSV ; séparateur :point-virgule, **jeu de caractères : Unicode utf-8**)

Attention : les logiciels de pointage retournent des tableaux de colonnes avec des entêtes (une à deux lignes) qu'il faudra par la suite retranscrire sous forme de listes (une liste par colonne) sans tenir compte des entêtes.

Voici une capture d'écran du fichier parabole.csv obtenu à l'aide de Regavi/Regressi ouvert sous Excel

| | A | B | C |
|---|---|---|---|
| 1 | t | x | y |
| 2 | s | m | m |
| 3 | 0 | -0,00280894 | 0 |
| 4 | 0,04 | 0,06460572 | 0,14325617 |
| 5 | 0,08 | 0,14044722 | 0,26684972 |
| 6 | 0,12 | 0,21347978 | 0,37639855 |
| 7 | 0,16 | 0,28651233 | 0,47190267 |
| 8 | 0,2 | 0,36235383 | 0,55336205 |
| 9 | 0,24 | 0,43538639 | 0,61796778 |
| 10 | 0,28 | 0,51403683 | 0,66571983 |
| 11 | 0,32 | 0,58426044 | 0,69380928 |
| 12 | 0,36 | 0,66291089 | 0,71347189 |
| 13 | 0,4 | 0,73875239 | 0,71347189 |
| 14 | 0,44 | 0,81459389 | 0,69661822 |
| 15 | 0,48 | 0,89043539 | 0,66010194 |
| 16 | 0,52 | 0,96627689 | 0,61796778 |
| 17 | 0,56 | 1,03930944 | 0,55336205 |
| 18 | 0,6 | 1,11515094 | 0,46909372 |
| 19 | 0,64 | 1,19099244 | 0,37358961 |
| 20 | 0,68 | 1,26964289 | 0,26123183 |
| 21 | 0,72 | 1,3398665 | 0,13482933 |
| 22 | | | |

Le même fichier ouvert sous Jupiter Notebook




2. Les cellules suivantes contiennent les lignes de code qui vous permettront d'afficher votre tableau de données sous forme de listes (une liste par colonne de votre tableau)

[1]:
```
# Chargement de la bibliothèque csv afin de pouvoir lire par la suite
# le fichier csv

import csv
```

[2]:
```
# création de la fonction appelée charge_fichier_csv() qui
# permettra de récupérer les données des colonnes d'un fichier.csv
# (ou fichier.txt pour le logiciel Avistep)
# Il faut préciser le délimiteur de colonnes utilisé dans le
# fichier .csv (ici c'est par défaut ";", pour les fichiers .txt d'Avistep
# c'est "\t")
# Il faut préciser le nombre de lignes d'en-tête N du fichier en tenant
# compte des lignes vides, par défaut N=0.

def charge_fichier_csv(fichier, delimiter =";",N=0):
```

(continues on next page)

(suite de la page précédente)

```
# ouverture du fichier .csv (ou fichier.txt pour avistep)

    with open(fichier, 'r', encoding='utf-8') as f :

# lecture du fichier à l'aide de la fonction csv.reader.

        rfichier = csv.reader(f, delimiter=delimiter)

# création et initialisation du tableau sous forme de liste qui recevra
# les listes de nombres réels correspondant aux colonnes

        tableau=[]

# le contenu d'une cellule est initialement lu comme une chaîne de
# caractères
# nous voulons obtenir des listes de nombres réels correspondant
# aux colonnes de notre tableau csv,
# donc :
#        - il ne faut pas prendre en compte les lignes
#          correspondant aux entêtes et les lignes vides
#        - il faut convertir les chaines de caractères en nombres
#          réels décimaux

# attention : les virgules des nombres décimaux doivent être
# remplacées par des points

# test permettant de sauter les lignes d'en-tête
        index_row=0                     # indice de la ligne = 0
        for row in rfichier:            # pour chaque ligne du fichier csv
            if index_row < N:
                index_row = index_row+1

# on parcourt chaque cellule d'une ligne du tableau csv

            else :
                for i in range (len(row)):

# Lors du parcours de la première ligne, on crée pour chaque cellule
# une liste vide qui contiendra par la suite les valeurs d'une colonne
# du fichier csv.
# Puis on l'ajoute au tableau
                    if len(tableau) <= i:
                        X = []
                        tableau.append(X)

# Pour chaque ligne, on ajoute à chaque liste créée précédemment
# les valeurs des cellules parcourues en les convertissant en
# nombre réel décimal et en évitant les erreurs liées
# souvent à des cellules vides (cas de Avistep)
                    try:
                        tableau[i].append(float(row[i].replace(",",'.')))
                    except ValueError:
                        print('erreur:contenu de cellule non numérique')
                        continue

        return (tableau)
```

Voici la fonction sans commentaire afin d'y voir un peu plus clair !

```
[3]: def charge_fichier_csv(fichier, delimiter=";",N=0):
    """
```

(continues on next page)

(suite de la page précédente)

```
    Charge un fichier csv et le renvoie sous forme de tableau

    :param: nom de fichier, délimiteur de cellules (par défaut ";"),
    nombre de lignes d'en-tête (en comptant les lignes vides)
    :returns: tableau des données
    """

    with open(fichier, 'r', encoding='utf-8') as f :
        rfichier = csv.reader(f, delimiter=delimiter)
        tableau=[]
        index_row=0
        for row in rfichier:
            if index_row < N:
                index_row = index_row+1
            else :
                for i in range (len(row)):
                    if len(tableau) <= i:
                        X = []
                        tableau.append(X)
                    try:
                        tableau[i].append(float(row[i].replace(",",'.')))
                    except ValueError:
                        print('erreur:contenu de cellule non numérique')
                        continue

        return (tableau)
```

[4]:
```
# Le début du chemin n'a pas besoin d'être spécifié si le fichier
# .csv se trouve dans le même dossier que ce fichier notebook

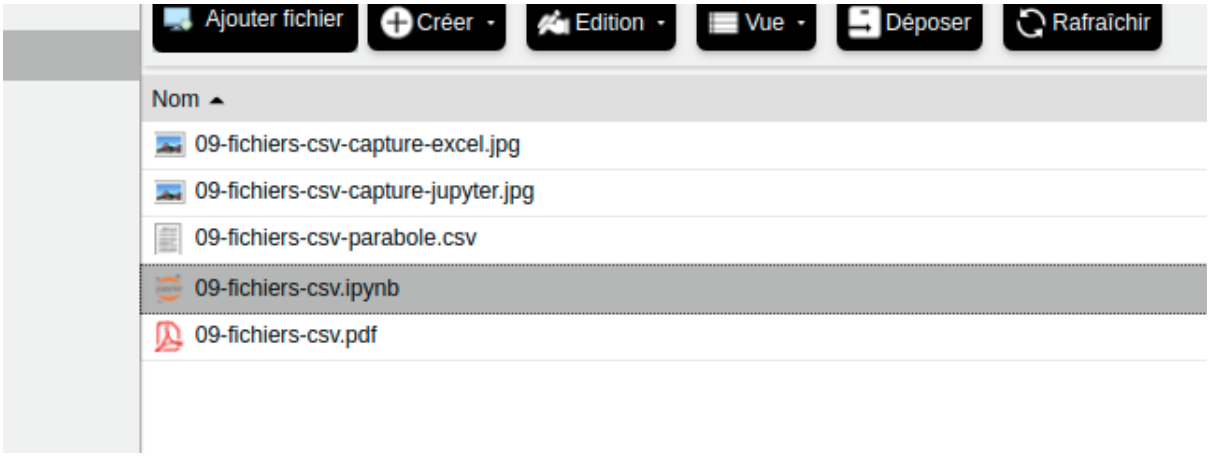
tableau = charge_fichier_csv('09-fichiers-csv-parabole.csv',delimiter=";",N=2)
t=tableau[0]
print(t)
x=tableau[1]
print(x)
y=tableau[2]
print(y)
```

```
[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]
[-0.002808944, 0.064605722, 0.140447222, 0.213479777, 0.286512333, 0.362353832, 0.
↪435386388, 0.514036832, 0.584260443, 0.662910887, 0.738752387, 0.814593887, 0.
↪890435386, 0.966276886, 1.039309442, 1.115150941, 1.190992441, 1.269642885, 1.
↪339866496]
[0.0, 0.143256166, 0.266849722, 0.376398555, 0.471902665, 0.553362054, 0.617967776,
↪ 0.665719832, 0.693809276, 0.713471887, 0.713471887, 0.69661822, 0.660101943, 0.
↪617967776, 0.553362054, 0.469093721, 0.37358961, 0.261231833, 0.134829333]
```

**WARNING : subtilité pour le jupyter ENT Nero 2018**

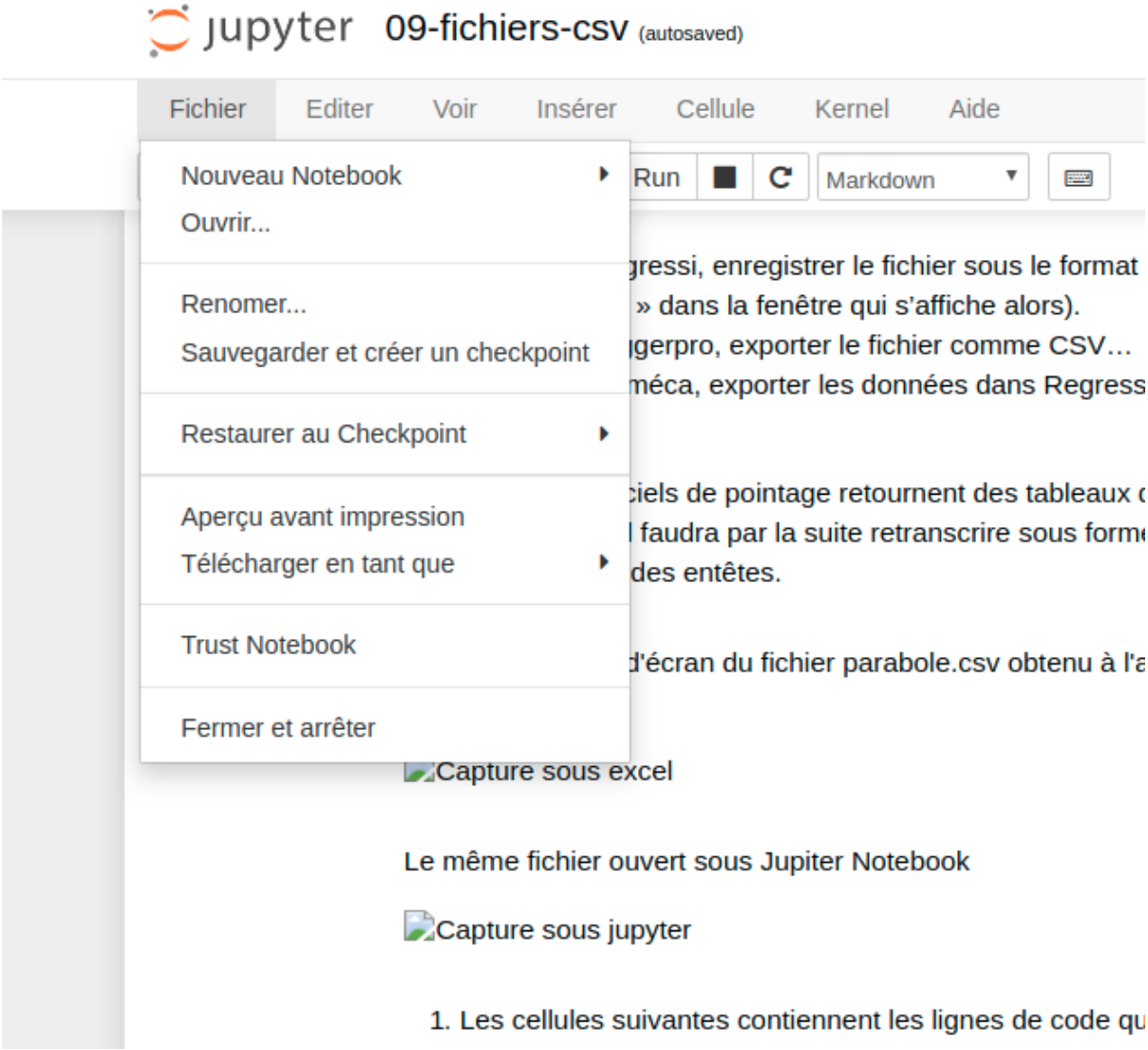
Même si vous avez mis vos fichiers CSV dans le même répertoire que votre fichier ipynb dans votre espace « Mes documents » de l'ENT, l'importation des données csv ne fonctionnera pas (en passant, l'affichage des images de ce notebook non plus, d'ailleurs..)

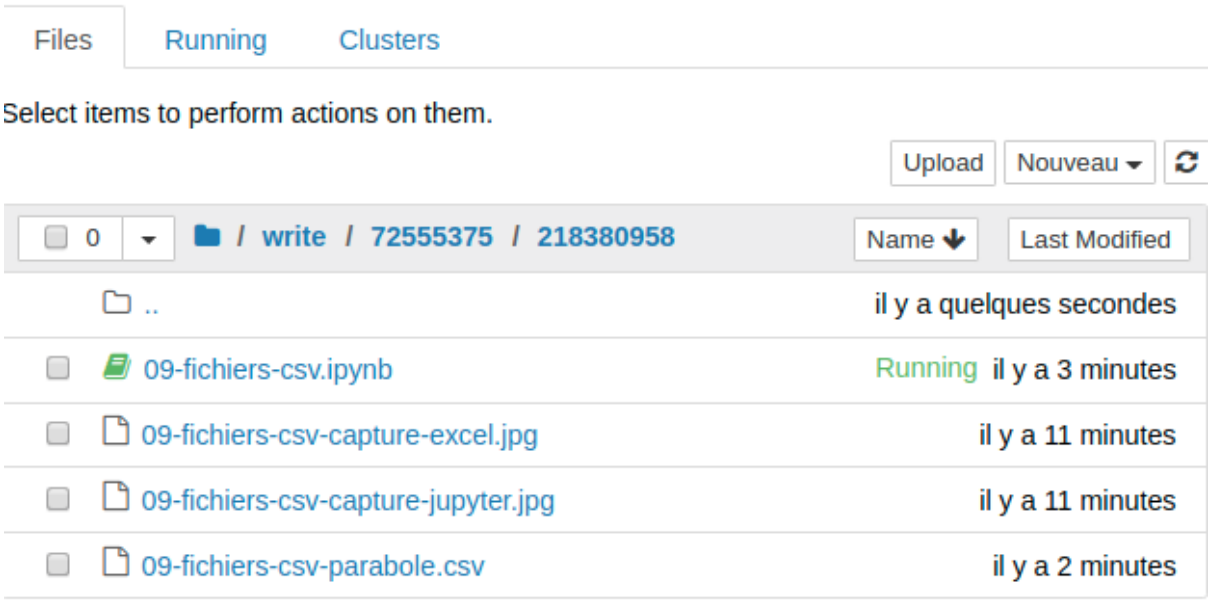
En fait sur cette version 2018, lors de l'exécution d'un notebook ipython, l'ENT copie ce fichier dans un répertoire temporaire vide… donc raté pour les fichers CSV.




Il existe une solution qui ressemble un peu à un bricolage, mais qui fonctionne en attendant 2019 :

- dans le notebook jupyter, cliquer sur « Fichier > Ouvrir »
- le répertoire de travail de votre notebook apparaît alors dans un onglet séparé
- vous pouvez alors y placer votre fichier csv avec le bouton « Upload »






A partir de la rentrée 2019, un autre mécanisme est possible pour charger des fichiers présents dans l'ENT, en utilisant une bibliothèque spécifique : NEROFS. Toutefois la méthode précédente fonctionne encore bien et est un peu plus simple.

[ ]:

### 2.1.10 Les graphiques (première partie)



`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Ceci est un petit tutoriel permettant de tracer des graphiques très simples mais suffisants dans le cadre du programme de physique-chimie. Pour plus d'informations, se référer au site : https://matplotlib.org/tutorials/index.html

Nous vous conseillons d'enregistrer ce fichier notebook sous un nom personnalisé afin de tester et modifier les lignes de codes à votre guise sans impacter la version originale.

Voici deux programmes (l'un très simple, l'autre plus complexe permettant d'afficher la caractéristique tension-intensité d'un conducteur ohmique à partir du tableau de valeurs suivantes :

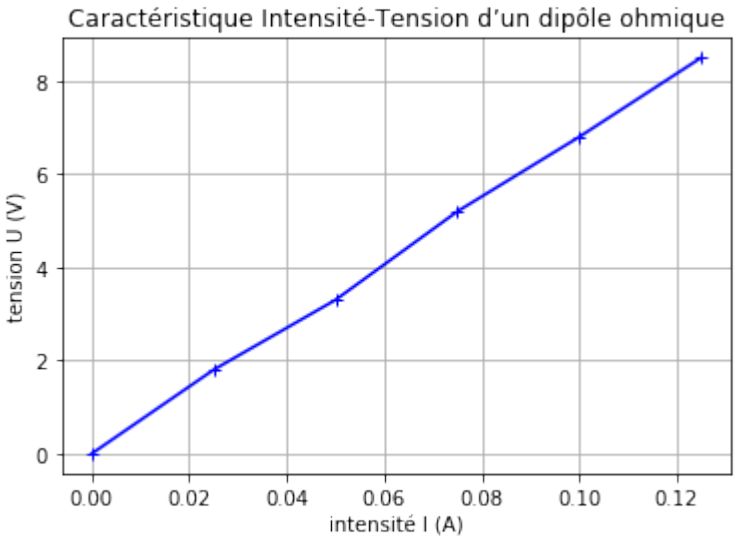
| I(mA) | 0 | 25 | 50 | 75 | 100 | 125 |
|---|---|---|---|---|---|---|
| U(V) | 0 | 1,8 | 3,3 | 5,2 | 6,8 | 8,5 |

**Dans un premier temps, vous allez exécuter les deux programmes fournis en observant les différences notables.**

**Dans un second temps, nous allons expliquer chacun de ces programmes**

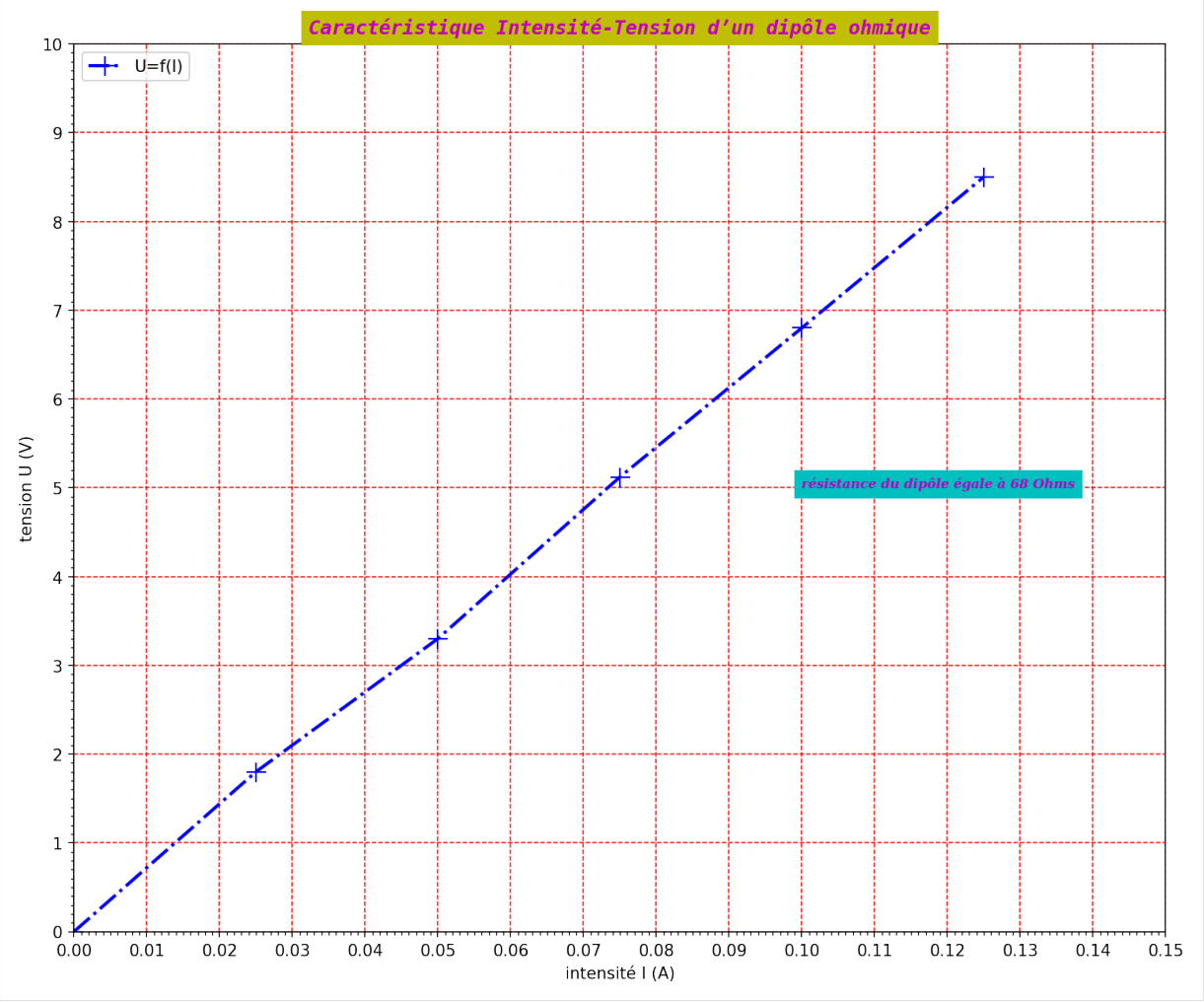
Premier programme (le plus simple)

[1]:
```
import matplotlib.pyplot as plt
%matplotlib inline
I=[0,25e-3,50e-3,75e-3,100e-3,125e-3]
U=[0,1.8,3.3,5.2,6.8,8.5]
plt.figure()
plt.plot(I,U,color='b', marker = '+')
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension d'un dipôle"
          " ohmique")
plt.show()
```




Deuxième programme (plus complexe)

[2]:
```
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
%matplotlib inline
I=[0,25e-3,50e-3,75e-3,100e-3,125e-3]
U=[0,1.8,3.3,5.12,6.8,8.5]
plt.figure("loi d'Ohm", figsize=(12,10), dpi=150)
plt.plot(I,U,color='b', marker = '+',markersize = 12,
         linestyle='-.', linewidth = 2, label='U=f(I)')
plt.legend(loc=2)
plt.xlim(0,0.150)
plt.ylim(0,10)
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.gca().xaxis.set_major_locator(MultipleLocator(0.01))
plt.gca().xaxis.set_minor_locator(MultipleLocator(0.001))
plt.gca().yaxis.set_major_locator(MultipleLocator(1))
plt.gca().yaxis.set_minor_locator(MultipleLocator(0.1))
plt.grid(color='r', linestyle='--', linewidth=0.75)
plt.title("Caractéristique Intensité-Tension d'un dipôle ohmique",
          fontsize=12,family='monospace',fontweight='bold',
          style='italic',color='m', backgroundcolor='y',
          horizontalalignment='center')
plt.text(0.100,5,'résistance du dipôle égale à 68 Ohms',
         fontsize=8, family='serif',fontweight='heavy',
         style='oblique',color='m',backgroundcolor='c', alpha=1)
plt.show()
```




**EXPLICATIONS DES PROGRAMMES**

Attention à bien exécuter les cellules de code suivantes les unes après les autres. Il est normal que rien ne s'affiche lors de l'exécution de certaines cellules.

#### 2.1.10.1 Import des bibliothèques utiles pour la création de graphiques

[3]:
```
# bibliothèque matplotlib.pyplot obligatoire
# pour la création de graphiques

import matplotlib.pyplot as plt

# %matplotlib inline permet d'afficher les graphiques
# matplotlib sous les cellules de code du notebook

%matplotlib inline

# bibliothèque numpy utile pour créer une liste de graduations

import numpy as np

# bibliothèque matplotlib.ticker utile ici uniquement pour
# graduer les axes à sa convenance avec la fonction MultipleLocator

from matplotlib.ticker import MultipleLocator
```

#### 2.1.10.2 Créer des listes contenant les valeurs du tableau

Pour créer un graphique sous Python, il faut tout d'abord créer un tableau de valeurs ou l'importer à partir d'un fichier .csv (voir tuto Comment importer les données numériques d'un tableur scientifique dans un programme python ?)

[4]:
```
# création d'une liste pour la grandeur portée en abscisse
# ici l'intensité est convertie en Ampère

I=[0,25e-3,50e-3,75e-3,100e-3,125e-3]

# création d'une liste pour la grandeur portée en ordonnée
# ici la tension

U=[0,1.8,3.3,5.2,6.8,8.5]
```

#### 2.1.10.3 Créer et paramétrer une fenêtre graphique

On utilise la méthode **plt.figure(num, figsize, dpi)** de la bibliothèque matplotlib.pyplot as plt.

Voici les principaux paramètres de cette méthode, il n'est pas obligatoire de tous les spécifier. Dans ce cas, ils prendront leur valeur par défaut.

- num : dénomination de la fenêtre graphique : nombre entier ou chaîne de caractères ; n'apparaît pas sur le graphique
- figsize =(x,y) : largeur x , hauteur y en pouces, valeur par défaut : [6.4, 4.8]
- dpi : résolution de la figure, par défaut : 100

**(voir ligne 5 du premier programme et ligne 6 du second programme)**

D'autres paramètres existent, pour plus d'informations :

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html#matplotlib.pyplot.figure

#### 2.1.10.4 Afficher les points ainsi que leur légende

Toujours à l'aide de la bibliothèque matplotlib.pyplot as plt :

1. Pour gérer l'affichage des points, on utilise la méthode **plt.plot(x, y, color, marker, markersize, linestyle, linewidth, label)**

Les paramètres présentés ici sont :

- x et y : grandeurs portées en abscisse et en ordonnée
- color : couleur des points et de la ligne les reliant

| code de la couleur | couleur |
|---|---|
| "b" | bleu |
| "g" | vert |
| "r" | rouge |
| "c" | cyan |
| "m" | magenta |
| "y" | jaune |
| "k" | noir |
| "w" | blanc |

Pour plus d'informations sur les couleurs :

https://matplotlib.org/tutorials/colors/colors.html#sphx-glr-tutorials-colors-colors-py

- marker : forme des points (marqueurs)

| code du marqueur | forme du marqueur |
|---|---|
| “.” | point |
| “,” | pixel |
| “o” | cercle |
| “+” | plus |
| “x” | fois |
| “v” | triangle vers le bas |
| “^” | triangle vers le haut |
| “<” | triangle vers la gauche |
| “>” | triangle vers la droite |
| “1” | étoile à trois branches dont l’une pointe vers le bas |
| “2” | étoile à trois branches dont l’une pointe vers le haut |
| “3” | étoile à trois branches dont l’une pointe vers la gauche |
| “4” | étoile à trois branches dont l’une pointe vers la droite |
| “s” | carré |
| “p” | pentagone |
| “*” | étoile |
| “h” | hexagone |
| “H” | hexagone |
| “D” | diamant |
| “d” | losange |
| “AltGr+6” | ligne verticale |
| “_” | ligne horizontale |

— markersize : taille du marqueur
— linestyle : style de ligne

| code du style | style de ligne |
|---|---|
| “-“ | ligne en trait plein |
| “–” | ligne en pointillé long |
| “-.” | ligne en pointillé mixte |
| “ :” | ligne en pointillé court |

— linewidth : épaisseur de la ligne
— label : permet de légender la courbe en créant une étiquette

D’autres paramètres existent. Pour plus d’informations :

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot

2. Pour faire apparaître l’étiquette (label) sur le graphique, il faut utiliser la méthode **plt.legend(loc)**.

Le paramètre loc permet de choisir l’emplacement de l’étiquette sur le graphique :

| Emplacement | Code d’emplacement |
|---|---|
| meilleur | 0 |
| en haut à droite | 1 |
| en haut à gauche | 2 |
| en bas à gauche | 3 |
| en bas à droite | 4 |
| droite | 5 |
| centre gauche | 6 |
| centre droit | 7 |
| centre bas | 8 |
| centre supérieur | 9 |
| centre | 10 |

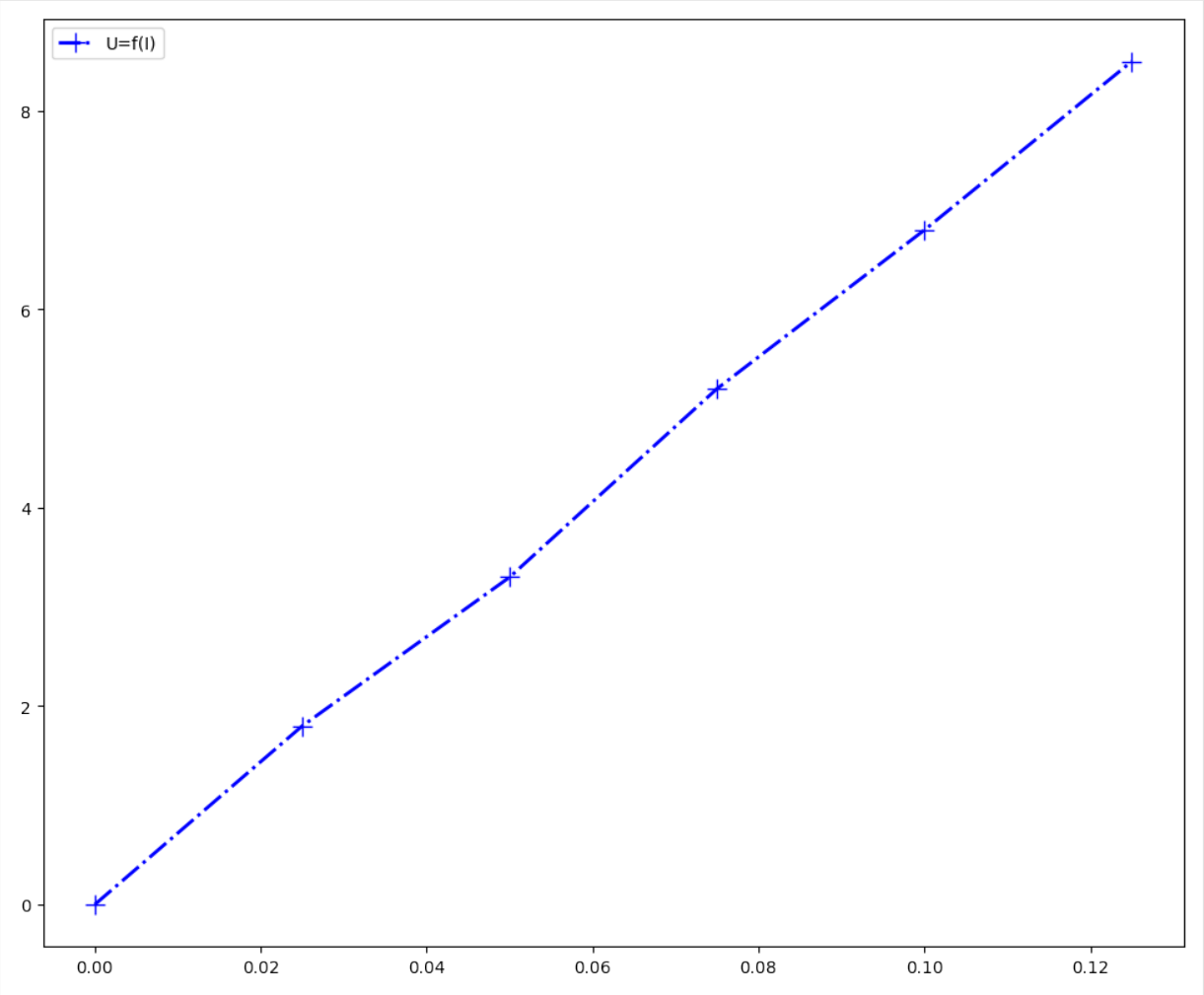
D’autres paramètres existent, notamment pour choisir la couleur , la police. . . .Pour plus d’informations :

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html

A vous maintenant de modifier les paramètres des trois lignes de code ci-dessous.

[5]:
```
plt.figure("loi d'Ohm", figsize=(12,10), dpi=100)
plt.plot(I,U,color='b', marker = '+',markersize = 12,
         linestyle='-.', linewidth = 2, label='U=f(I)')
plt.legend(loc=2)
```

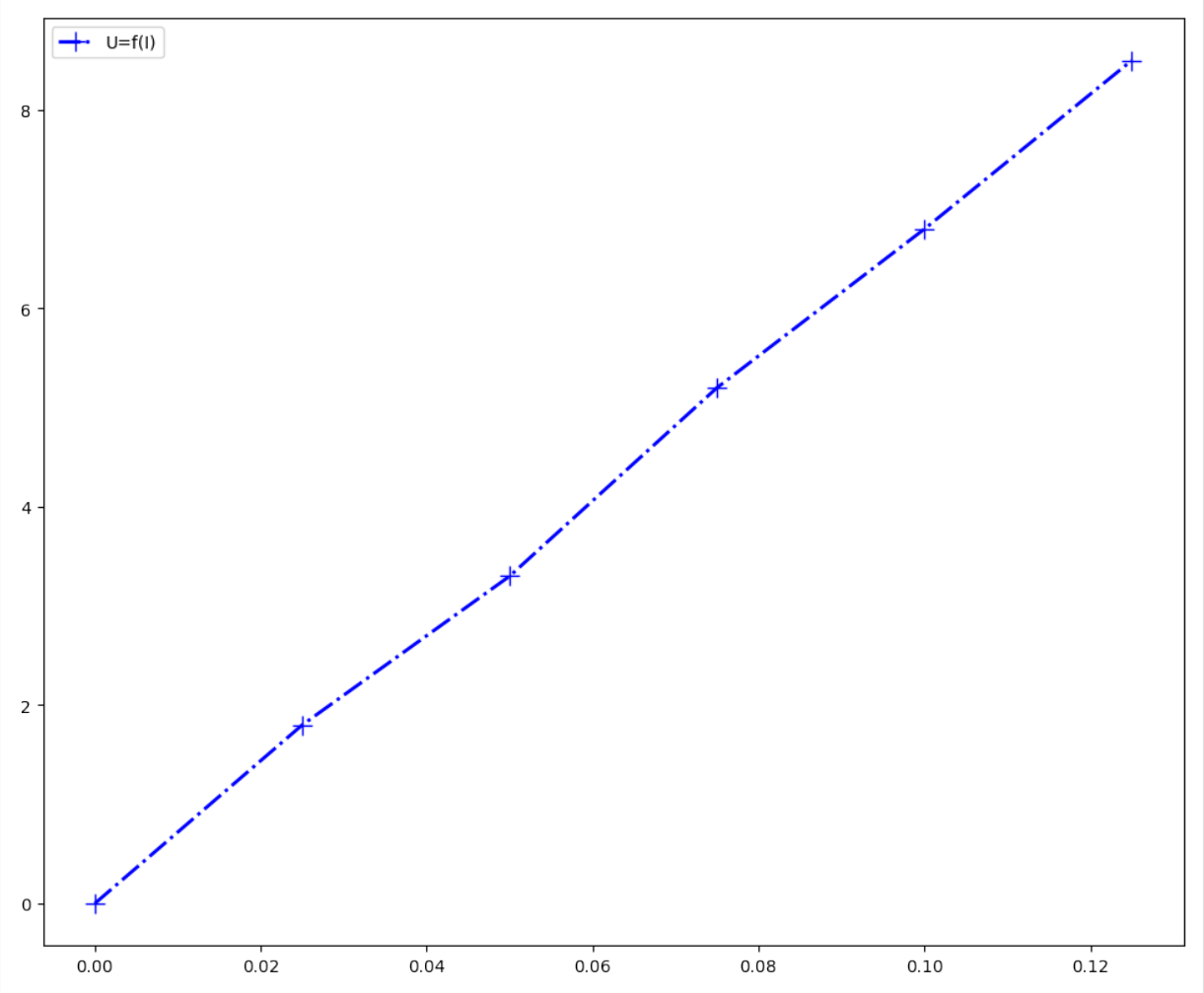
[5]:
```
<matplotlib.legend.Legend at 0x7f965c230080>
```




Une autre possibilité de codage plus simple pour un même résultat :

[6]:
```
plt.figure("loi d'Ohm", figsize=(12,10), dpi=100)
plt.plot(I,U,'b+-.',markersize = 12, linewidth = 2, label='U=f(I)')
plt.legend(loc=2)
```

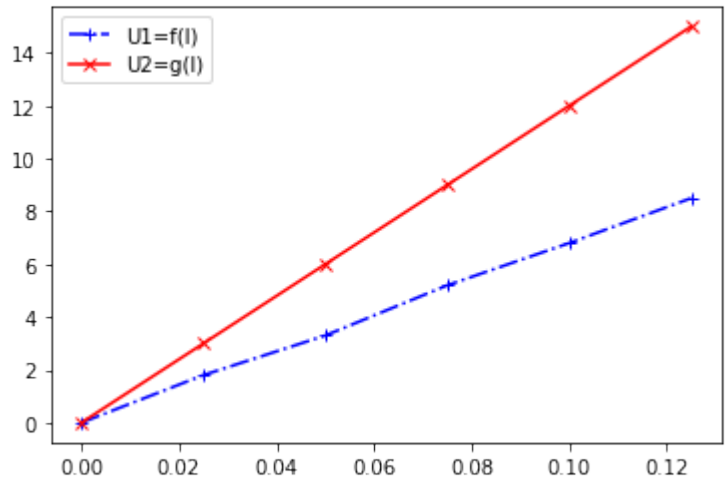
[6]:
```
<matplotlib.legend.Legend at 0x7f9620f34f98>
```




Et si l'on veut afficher deux courbes sur le même graphique ?

Première possibilité :

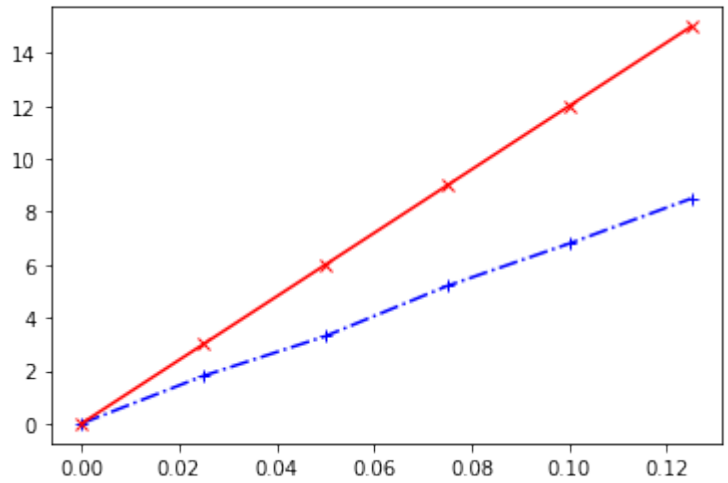
[7]:
```
I=[0,25e-3,50e-3,75e-3,100e-3,125e-3]
U1=[0,1.8,3.3,5.2,6.8,8.5]
U2 = [0,3,6,9,12,15]
plt.figure()
plt.plot(I,U1,'b+-.',label='U1=f(I)')
plt.plot(I,U2,'rx-',label='U2=g(I)')
plt.legend(loc=2)
plt.show()
```




Deuxième possibilité : La méthode plt.plot() renvoie une liste de courbes. Il est donc possible d'afficher deux courbes simultanément avec un seul appel de la methode plt.plot() comme dans l'exemple ci-dessous.

Remarque utile pour comprendre le prochain tutoriel sur les animations de graphique : L'indice de la première courbe est 0, celui de la deuxième est 1…

[8]:
```
I=[0,25e-3,50e-3,75e-3,100e-3,125e-3]
U1=[0,1.8,3.3,5.2,6.8,8.5]
U2 = [0,3,6,9,12,15]
plt.figure()
plt.plot(I,U1,'b+-.',I,U2,'rx-')
plt.show()
```




#### 2.1.10.5 Configurer les axes du graphique et faire apparaître le quadrillage

Toujours à l'aide de la bibliothèque matplotlib.pyplot as plt :

1. Pour délimiter les valeurs minimales et maximales des axes, il y a deux possibilités :
   — soit on utilise la méthode : **plt.axis([xmin,xmax,ymin,ymax])**
   — soit on utilise les méthodes : **plt.xlim(xmin,xmax)** et **plt.ylim(ymin,ymax)**
2. Pour légender les axes, on utilise les méthodes : **plt.xlabel(« légende1 »)** et **plt.ylabel(« légende2 »)**
3. La graduation des axes se fait par défaut. Pour graduer les axes à votre convenance, voici deux méthodes :
   **Première méthode :**
   — appeler la bibliothèque numpy : **import numpy as np**
   — utiliser les méthodes :

- **plt.xticks(np.arange(xmin, xmax, valeur d'une graduation))**
- **plt.yticks(np.arange(ymin, ymax, valeur d'une graduation))**

**Deuxième méthode :**

- appeler la fonction MultipleLocator de la bibliothèque matplotlib.ticker : **from matplotlib.ticker import MultipleLocator**
- utiliser les méthodes pour les graduations majeures :
  - **plt.gca().xaxis.set_major_locator(MultipleLocator(valeur d'une graduation))**
  - **plt.gca().yaxis.set_major_locator(MultipleLocator(valeur d'une graduation))**
- utiliser les méthodes pour les graduations mineures :
  - **plt.gca().xaxis.set_minor_locator(MultipleLocator(valeur d'une graduation))**
  - **plt.gca().yaxis.set_minor_locator(MultipleLocator(valeur d'une graduation))**

4. Pour faire apparaître le quadrillage sur les graduations majeures ou par défaut, on utilise la méthode : **plt.grid(color, linestyle, linewidth)**.

Les paramètres du quadrillage mentionnés ci-dessus sont les suivants :

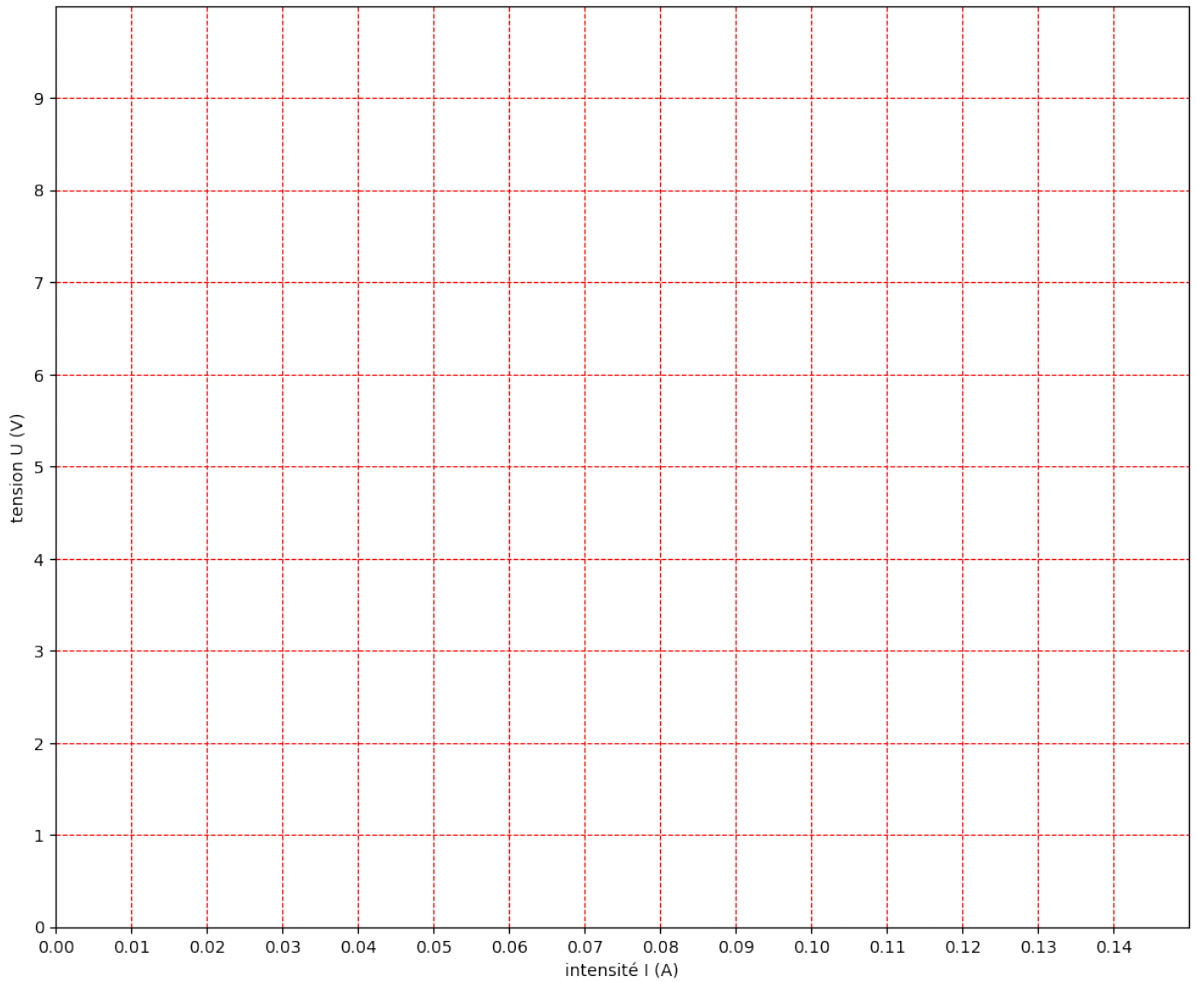
- color : la couleur des lignes (codes : cf. tableau ci-dessus)
- linestyle : le style des lignes (codes : cf. tableau ci-dessus)
- linewidth : l'épaisseur des lignes

D'autres paramètres existent. Pour plus d'informations :

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.grid.html

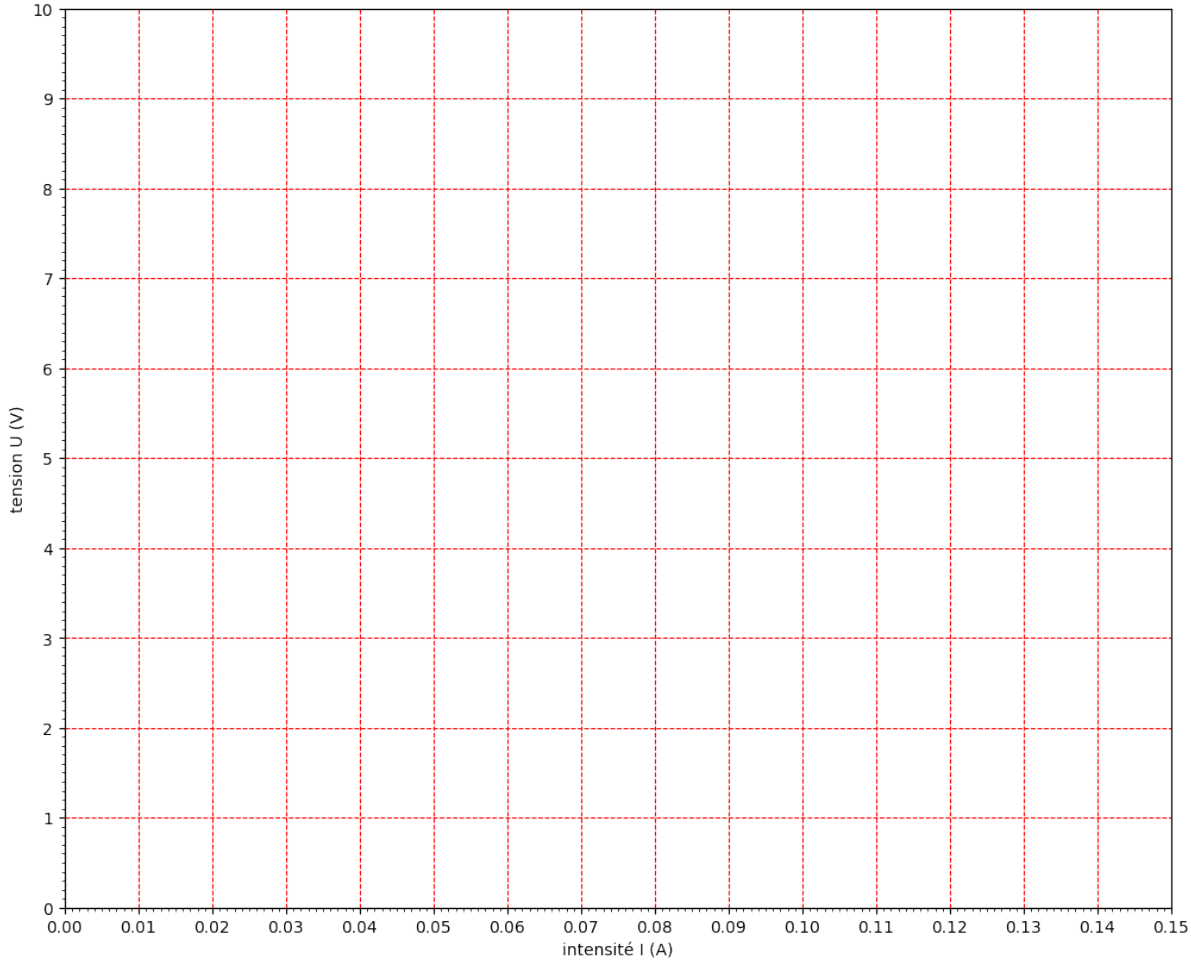
Premier exemple avec les méthodes plt.axis(), plt.xticks() et plt.yticks() :

[9]:
```
plt.figure(1, figsize=(12,10), dpi=100)
plt.axis([0,0.150,0,10])
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.yticks(np.arange(0,10,1))
plt.xticks(np.arange(0,0.150,0.01))
plt.grid(color='r', linestyle='--', linewidth=0.75)
```



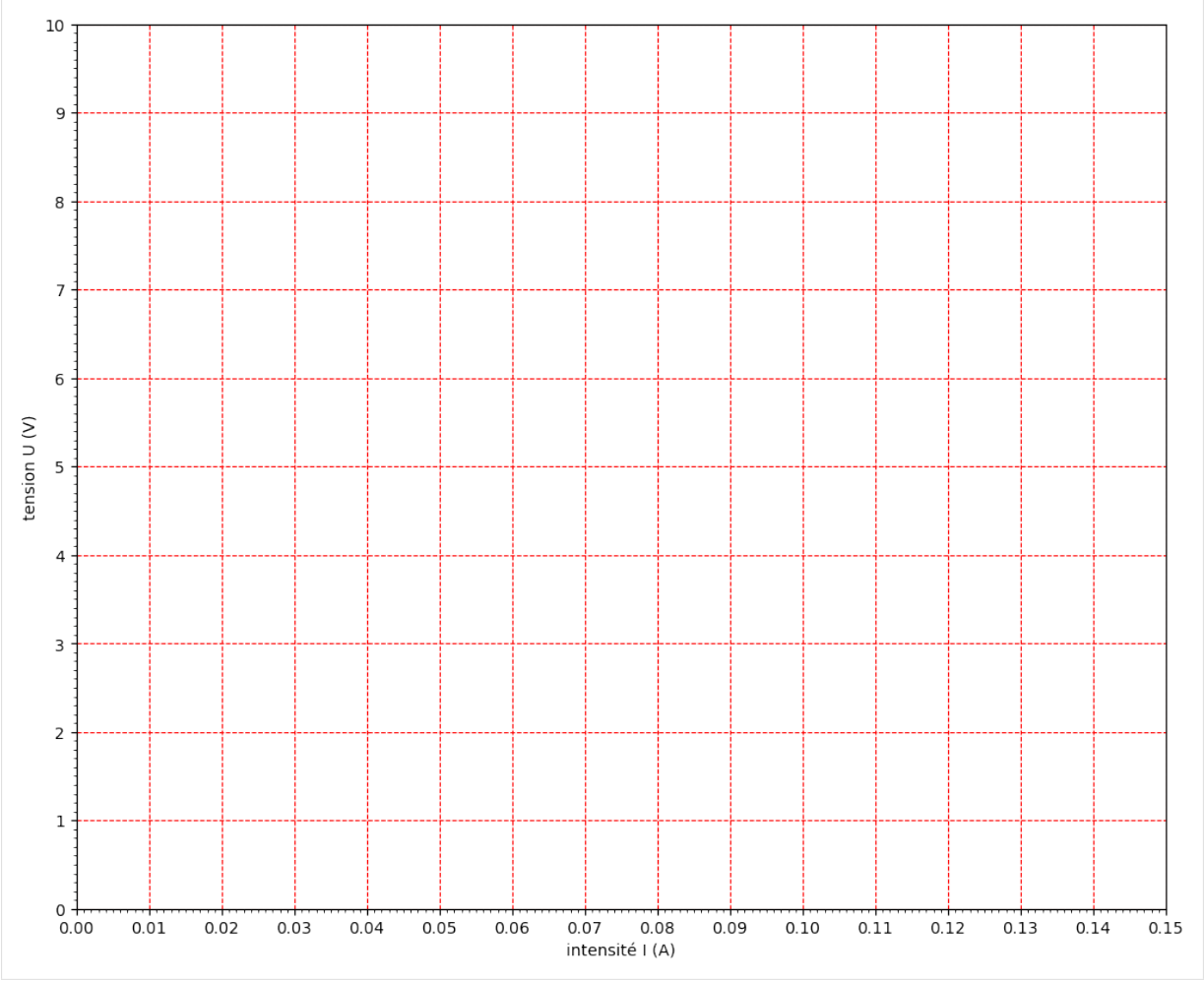

Deuxième exemple avec les méthodes plt.gca().xaxis.set_major_locator(MultipleLocator()) etc. . .

[10]:
```
plt.figure(2, figsize=(12,10), dpi=100)
plt.axis([0,0.150,0,10])
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.gca().xaxis.set_major_locator(MultipleLocator(0.01))
plt.gca().xaxis.set_minor_locator(MultipleLocator(0.001))
plt.gca().yaxis.set_major_locator(MultipleLocator(1))
plt.gca().yaxis.set_minor_locator(MultipleLocator(0.1))
plt.grid(color='r', linestyle='--', linewidth=0.75)
```




Troisième exemple avec les méthodes plt.xlim() et plt.ylim() qui donne un résultat identique au précédent :

[11]:
```
plt.figure(3, figsize=(12,10), dpi=100)
plt.xlim(0,0.150)
plt.ylim(0,10)
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.gca().xaxis.set_major_locator(MultipleLocator(0.01))
plt.gca().xaxis.set_minor_locator(MultipleLocator(0.001))
plt.gca().yaxis.set_major_locator(MultipleLocator(1))
plt.gca().yaxis.set_minor_locator(MultipleLocator(0.1))
plt.grid(color='r', linestyle='--', linewidth=0.75)
```




#### 2.1.10.6 Ecrire un titre et/ou un texte sur le graphique

On utilise la méthode **plt.title(« Titre »,...)** et/ou la méthode **plt.text(x,y, »Texte », ...)** de la bibliothèque matplotlib.pyplot as plt.

x, y sont les coordonnées du texte sur le graphique

Voici quelques paramètres pouvant être ajoutés pour modifier le texte affiché :

- fontsize : la taille de la police de caractères
- family : le type de police (“serif”, “sans-serif”, “monospace”).
- fontweight : l’épaisseur de la police (“normal”, “bold”, “heavy”, “light”, “ultrabold”, “ultralight”).
- style : le style de la police (“normal”, “italic”, “oblique”).
- color : la couleur de la police.
- backgroundcolor : la couleur du fond.
- horizontalalignment : permet de centrer le texte (“left”, “center”, “right”). Attention : “left” veut dire que c’est la partie gauche du texte qui est positionnée au centre du graphique donc cette commande décale le texte vers la droite.
- alpha : permet de moduler la transparence du texte (0 : transparent ; 1 : opaque)

[12]:
```
plt.xlim(0,0.150)
plt.ylim(0,10)

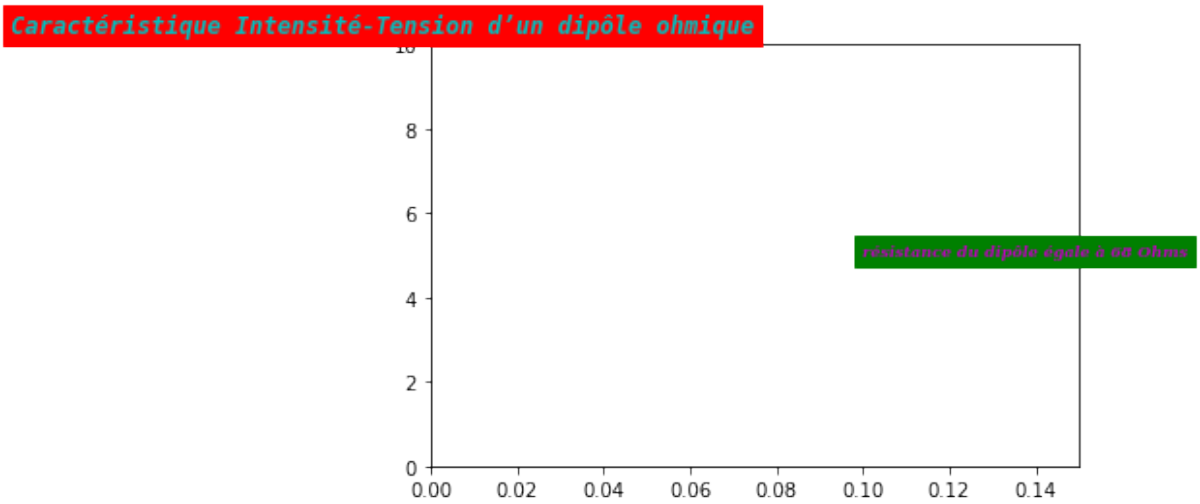
plt.title("Caractéristique Intensité-Tension d'un dipôle ohmique",
          fontsize=12,family='monospace',fontweight='bold',
          style='italic',color='c',backgroundcolor='r',
          horizontalalignment='right')
```

(continues on next page)

(suite de la page précédente)

```
plt.text(0.100,5,'résistance du dipôle égale à 68 Ohms',
         fontsize=8,family='serif',fontweight='heavy',
         style='oblique',color='m',backgroundcolor='g',alpha=1)
```

[12]:
```
Text(0.1, 5, 'résistance du dipôle égale à 68 Ohms')
```




#### 2.1.10.7 Afficher la fenêtre graphique

On utilise la méthode **plt.show()** de la bibliothèque matplotlib.pyplot as plt en fin de programme afin d'afficher la fenêtre graphique réalisée précédemment.

### 2.1.11 Les graphiques (deuxième partie)



`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Dans cette deuxième partie, nous allons voir comment créer des graphiques multiples et des vecteurs.

Pour plus d'informations, se référer au site : https://matplotlib.org/tutorials/index.html

#### 2.1.11.1 Afficher plusieurs graphiques simultanément

Pour cela, on utilise la méthode **plt.subplot(nombre de lignes, nombre de colonnes, index)** de la bibliothèque matplotlib.pyplot as plt.

Cette méthode permet de créer un emplacement en divisant la fenêtre graphique en un nombre donné de lignes et de colonnes indiqué entre parenthèses. L'index permet de renseigner la position de l'emplacement dans la fenêtre graphique. Il commence à 1 dans le coin supérieur gauche de la fenêtre graphique et augmente vers la droite puis vers le bas.

Exemple : pour une fenêtre graphique divisée en 2 lignes et 2 colonnes, les emplacements sont indexées comme suit :

**1 2**

**3 4**

Après avoir défini l'emplacement, il suffit de coder directement les caractéristiques du graphique qui apparaîtra dans l'emplacement en question.

Pour ajuster les positions des côtés des graphiques et les espacements entre ceux-ci, on utilise la méthode :

**plt.gcf().subplots_adjust(left = 0.125, bottom = 0.1, right = 0.9, top = 0.9, wspace = 0.2, hspace = 0.2)** de la bibliothèque matplotlib.pyplot as plt.

Les paramètres utilisés sont les suivants (les valeurs données ci-dessous sont les valeurs par défaut) :
- left = 0.125 : position du côté gauche du graphique
- right = 0.9 : position du côté droit du graphique
- bottom = 0.1 : position du côté bas du graphique
- top = 0.9 : position côté haut du graphique
- wspace = 0.2 : espacement horizontal entre deux graphiques
- hspace = 0.2 : espacement vertical entre deux graphiques

Toutefois, si la taille de la fenêtre graphique n'est pas cohérente avec les valeurs des paramètres, l'affichage sera modifié.

Prenons l'exemple de la chute libre d'une balle sans vitesse initiale afin d'illustrer ces différentes méthodes. Voici trois programmes permettant d'afficher les coordonnées verticales de la position, de la vitesse et de l'accélération d'une balle en fonction du temps sur trois graphiques simultanément. Selon le programme, les graphiques sont positionnés différemment.

#### 2.1.11.1.1 Premier programme

[1]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

t=np.arange(0,0.65,0.05)
z=-4.9*t**2+2
vz=-9.8*t
az=-9.8+0*t

# création de la fenêtre graphique 1

plt.figure(1,figsize=(10,12))
plt.gcf().subplots_adjust(left = 0.125, bottom = 0.2, right = 1.5,
                          top = 0.9, wspace = 0.5, hspace = 0)

# division de la fenêtre graphique en 1 ligne, 3 colonnes,
# graphique en position 1
# puis caractéristiques de ce graphique

plt.subplot(1,3,1)
plt.plot(t,z,color='b', marker = '+')
plt.title('position')
plt.grid()
plt.xlabel('t(s)')
plt.ylabel('z(m)')
plt.ylim(0)


# division de la fenêtre graphique en 1 ligne, 3 colonnes,
# graphique en position 2
# puis caractéristiques de ce graphique

plt.subplot(1,3,2)
plt.plot(t,vz,color='b', marker = '+')
```

(continues on next page)

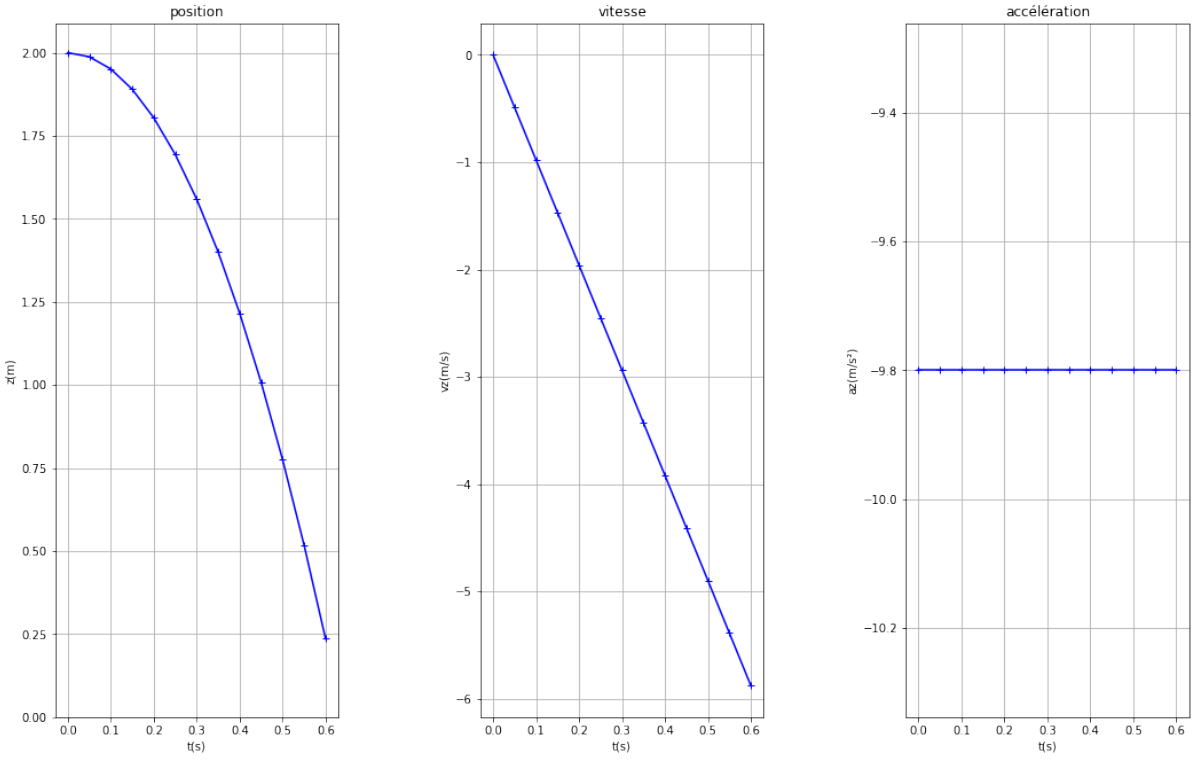
(suite de la page précédente)

```
plt.title ('vitesse')
plt.grid()
plt.xlabel('t(s)')
plt.ylabel('vz(m/s)')


# division de la fenêtre graphique en 1 ligne, 3 colonnes,
# graphique en position 3
# puis caractéristiques de ce graphique

plt.subplot(1,3,3)
plt.plot(t,az,color='b', marker = '+')
plt.title('accélération')
plt.grid()
plt.xlabel('t(s)')
plt.ylabel('az(m/s²)')

plt.show()
```




#### 2.1.11.1.2 Deuxième programme

[2]:
```
# création de la fenêtre graphique 2

plt.figure(2,figsize=(10,12))
plt.gcf().subplots_adjust(left = 0.2, bottom = 0.2, right = 1.5,
                          top = 0.9, wspace = 0, hspace = 0.5)

# division de la fenêtre graphique en 3 lignes, 1 colonne,
# graphique en position 1
# puis caractéristiques de ce graphique

plt.subplot(3,1,1)
```

(continues on next page)

(suite de la page précédente)

```
plt.plot(t,z,color='b', marker = '+')
plt.title('position')
plt.grid()
plt.xlabel('t(s)')
plt.ylabel('z(m)')
plt.ylim(0)

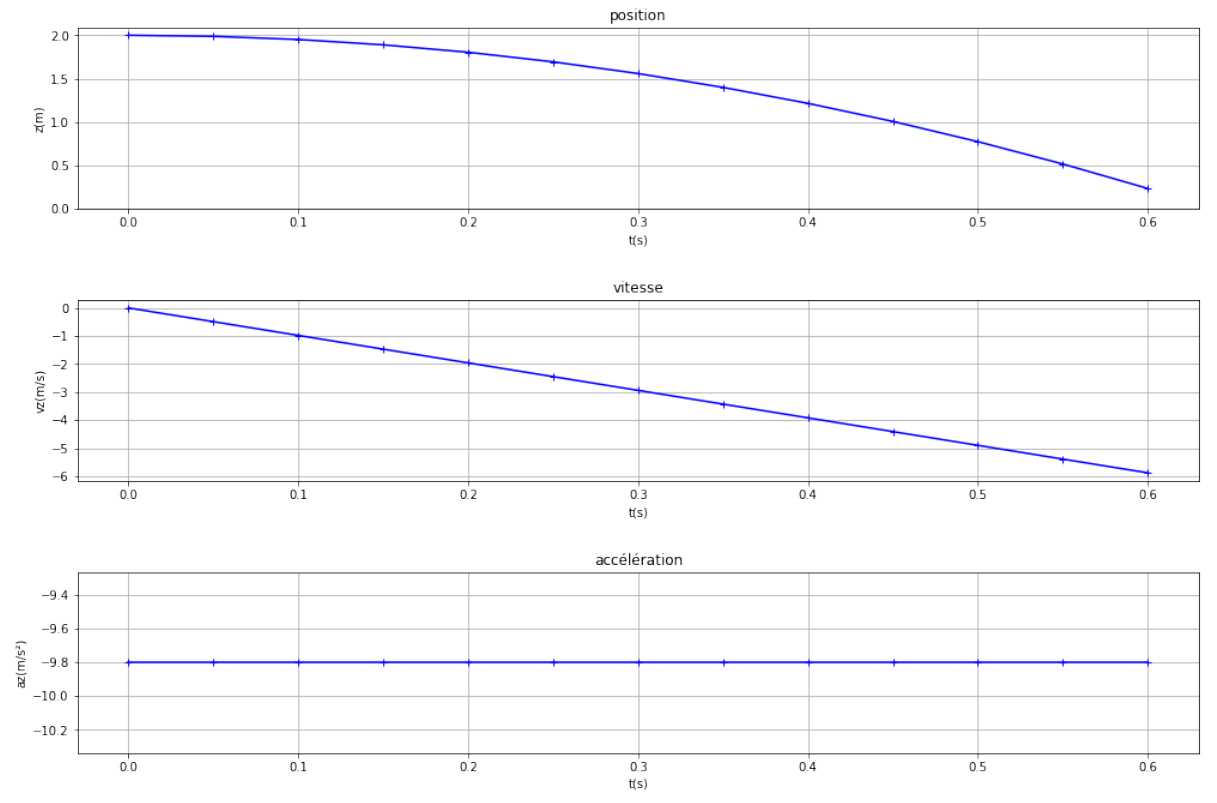

# division de la fenêtre graphique en 3 lignes, 1 colonne,
# graphique en position 2
# puis caractéristiques de ce graphique

plt.subplot(3,1,2)
plt.plot(t,vz,color='b', marker = '+')
plt.title ('vitesse')
plt.grid()
plt.xlabel('t(s)')
plt.ylabel('vz(m/s)')


# division de la fenêtre graphique en 3 lignes, 1 colonne,
# graphique en position 3
# puis caractéristiques de ce graphique

plt.subplot(3,1,3)
plt.plot(t,az,color='b', marker = '+')
plt.title('accélération')
plt.grid()
plt.xlabel('t(s)')
plt.ylabel('az(m/s$^2$)')

plt.show()
```



#### 2.1.11.1.3 Troisième programme

[3]:
```
# création de la fenêtre graphique 3

plt.figure(3,figsize=(10,12))
plt.gcf().subplots_adjust(left = 0.2, bottom = 0.2, right = 1.5,
                          top = 0.8, wspace = 0.5, hspace = 0.5)

# division de la fenêtre graphique en 1 ligne, 2 colonnes,
# graphique en position 1
# puis caractéristiques de ce graphique

plt.subplot(1,2,1)
plt.plot(t,z,color='b', marker = '+')
plt.title('position')
plt.grid()
plt.xlabel('t(s)')
plt.ylabel('z(m)')
plt.ylim(0)

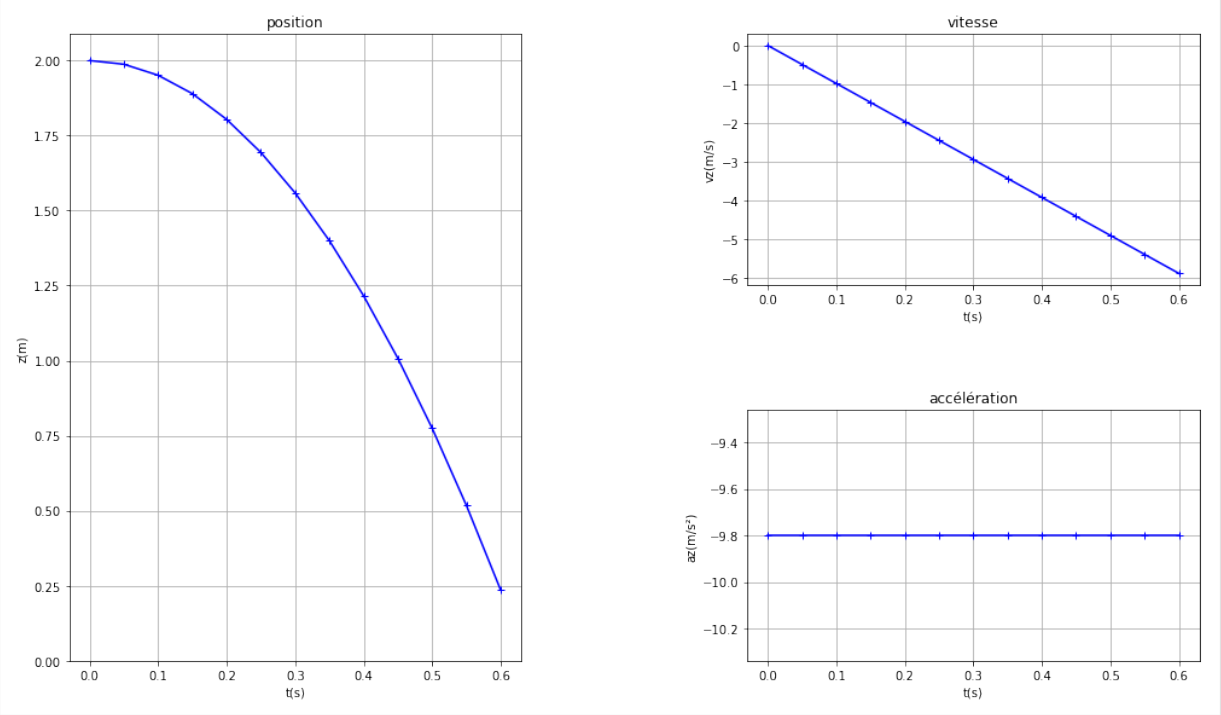

# division de la fenêtre graphique en 2 lignes, 2 colonnes,
# graphique en position 2
# puis caractéristiques de ce graphique

plt.subplot(2,2,2)
plt.plot(t,vz,color='b', marker = '+')
plt.title ('vitesse')
plt.grid()
plt.xlabel('t(s)')
plt.ylabel('vz(m/s)')


# division de la fenêtre graphique en 2 lignes, 2 colonnes,
# graphique en position 4
# puis caractéristiques de ce graphique

plt.subplot(2,2,4)
plt.plot(t,az,color='b', marker = '+')
plt.title('accélération')
plt.grid()
plt.xlabel('t(s)')
plt.ylabel('az(m/s$^2$)')

plt.show()
```




### 2.1.11.2 Afficher des vecteurs sur un graphique

Pour cela, on utilise la méthode de la bibliothèque matplotlib.pyplot as plt

**plt.arrow(x, y, composante_selon_x, composante_selon_y, facecolor="r", edgecolor="r", width=0.008, head_width=0.02, length_includes_head=True)**

Les paramètres utilisés sont :

- x, y : coordonnées de l'origine du vecteur dans la base utilisée dans le graphique.
- composante_selon_x, composante_selon_y : composantes (coordonnées) du vecteur
- facecolor (fc) : couleur de remplissage du vecteur
- edgecolor (ec) : couleur du contour du vecteur
- width : largeur du corps du vecteur (valeur par défaut : 0.001)
- head_width : largeur de la tête du vecteur (par défaut : 3 * width)
- head_length : longueur de la tête du vecteur (par défaut : 1.5 * head_width)
- length_includes_head : True si la tête doit être comptée dans le calcul de la longueur du vecteur (par défaut : False)

Prenons l'exemple d'un tir parabolique d'une balle et voyons comment tracer les vecteurs vitesse pour chaque position de la balle.

[4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

t=np.arange(0,1.05,0.05)

z=-4.9*t**2+5*t
vz=-9.8*t+5
x=3*t
vx=3+0*t


plt.figure(4,figsize=(17,7))

plt.plot(x,z,color='b', marker = '+')
```

(continues on next page)

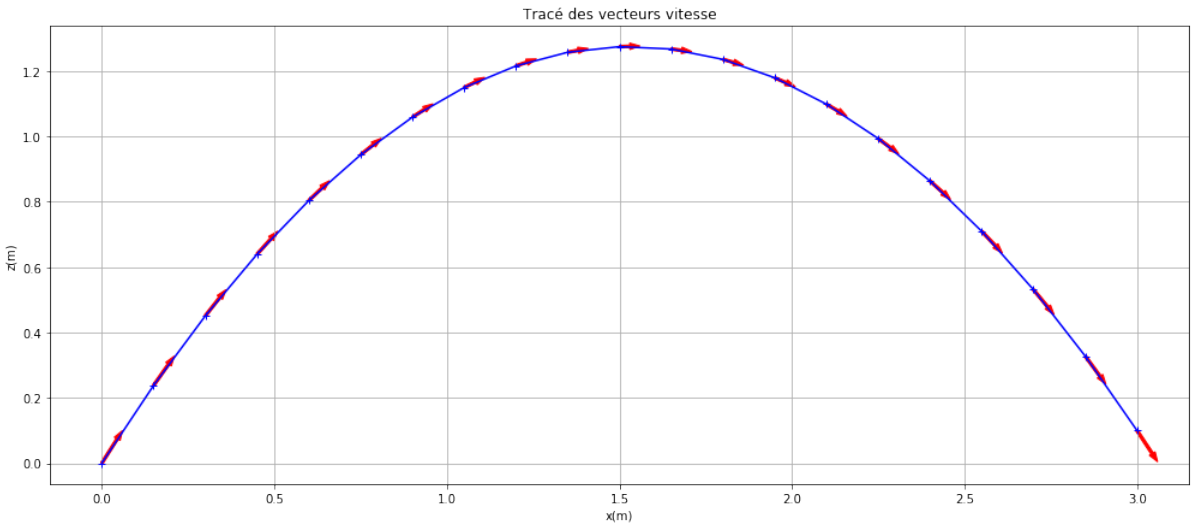
(suite de la page précédente)

```
plt.title('Tracé des vecteurs vitesse')
plt.grid()
plt.xlabel('x(m)')
plt.ylabel('z(m)')

for i in range (len (z)) :
    plt.arrow(x[i],z[i],0.02*vx[i],0.02*vz[i],facecolor='r',
              edgecolor='r',width=0.008,head_width=0.02,
              length_includes_head=True)

# ici les coordonnées vx et vy sont multipliées par 0.02 afin
# d'adapter l'échelle de longueur du vecteur.

plt.show()
```




### 2.1.12 Les modélisations



`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Dans ce tutoriel, on apprendra à modéliser un nuage de points.

Reprenons l'exemple de la loi d'Ohm (Tutoriel sur le tracé d'un graphique $1^{re}$ partie) et modélisons la courbe obtenue.

Affichons la courbe obtenue avant modélisation :

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# tableaux numpy impératifs pour réaliser le calcul (vectorisé) de
# la tension modélisée Umodel lors de la modélisation ultérieure

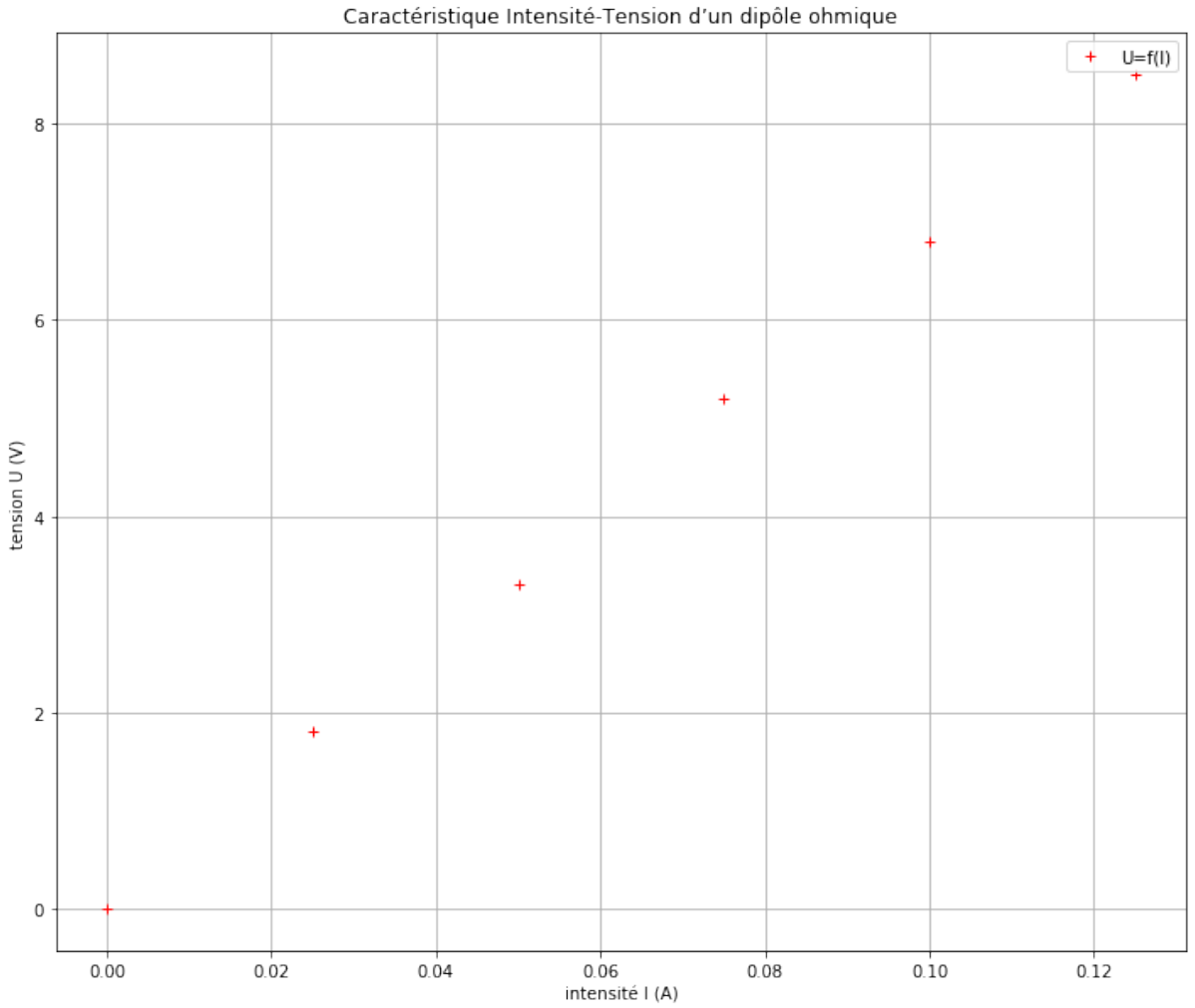
I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
U=np.array([0,1.8,3.3,5.2,6.8,8.5])

fig = plt.figure(figsize=(12,10))
```

(continues on next page)

(suite de la page précédente)

```
plt.plot(I,U,'r+',label='U=f(I)')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension d'un dipôle ohmique")
plt.show()
```




#### 2.1.12.1 Première partie : Modélisation à l'aide de la fonction polyfit de la bibliothèque numpy

Si le nuage de points est modélisable par une fonction polynomiale, on peut utiliser la fonction **np.polyfit( x , y , deg)** de la bibliothèque numpy as np.

Le paramètre deg correspond au degré du polynôme.

La fonction polyfit retourne un tableau numpy à une dimension de coefficients p qui minimisent l'erreur à l'aide de la méthode des moindres carrés dans l'ordre : deg , deg-1 ,... 0

$P(x) = p[0] \times x^{deg} + p[1] \times x^{deg-1} ... + p[deg]$

D'autres paramètres existent. Pour plus d'informations :

https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.polyfit.html

Dans le cadre du programme de physique-chimie au lycée, on a besoin au plus de polynômes du second degré.

- Pour un polynôme de degré 2 : $P(x) = p[0] \times x^2 + p[1] \times x + p[2]$
- Pour un polynôme de degré 1 : $P(x) = p[0] \times x + p[1]$

— Pour un polynôme de degré 0 : $P(x) = p[0]$

Modélisons maintenant la courbe obtenue par une droite (polynôme de degré 1) et affichons les coefficients :

[2]:
```
coeff=np.polyfit(I, U,1)
print(coeff[0],coeff[1])
```

```
67.88571428571427 0.023809523809522625
```

Avec une décimale :

[3]:
```
print ('{0:.1f}'.format(coeff[0]),'{0:.1f}'.format(coeff[1]))
```

```
67.9 0.0
```

Créons l'équation de la droite modélisée grâce à ces coefficients et affichons cette équation ainsi que son tableau de valeurs :

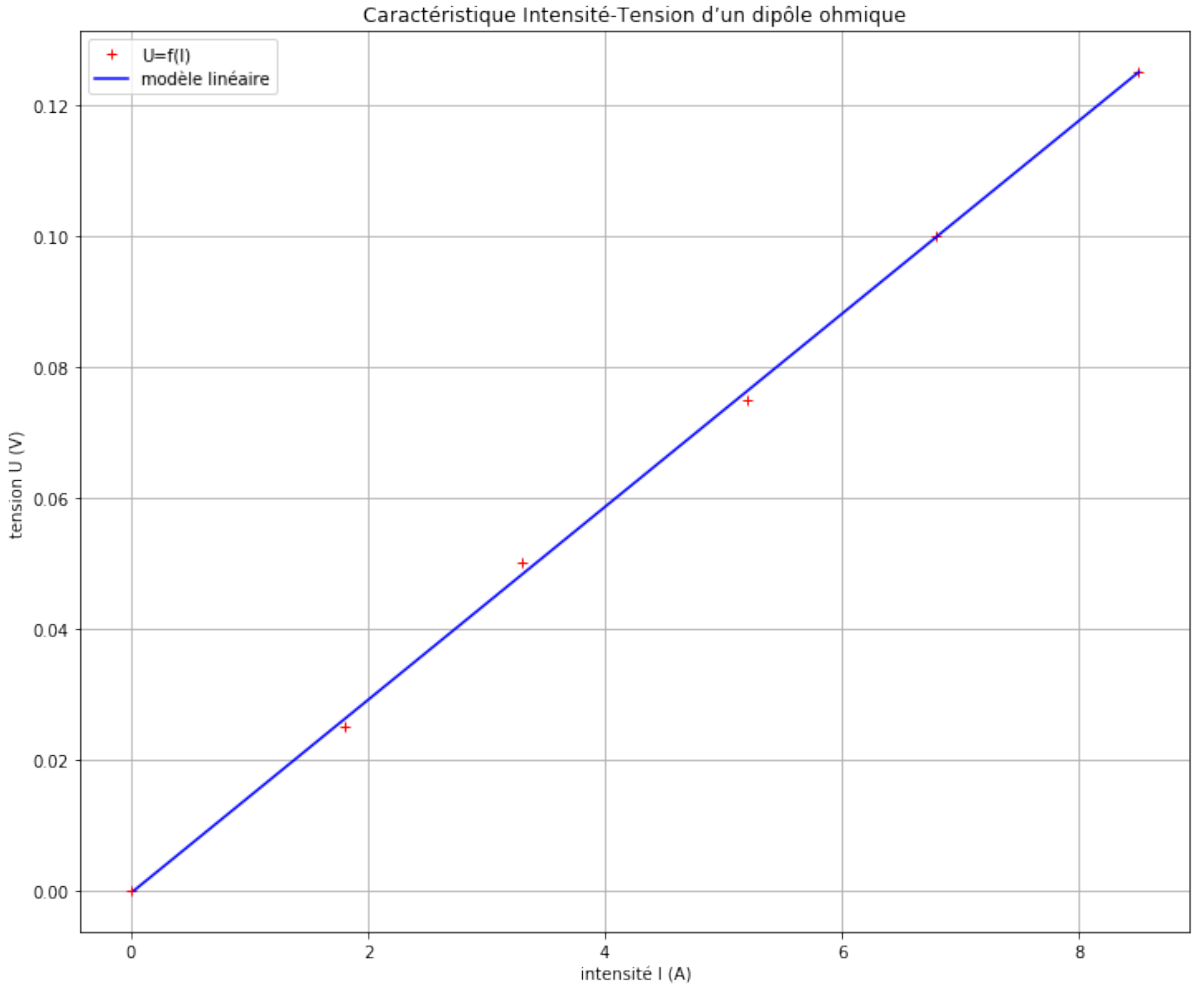
[4]:
```
Umodel = coeff[0]*I+coeff[1]
print('U={0:.1f}'.format(coeff[0]),'xI+{0:.1f}'.format(coeff[1]))
print(Umodel)
```

```
U=67.9 xI+0.0
[0.02380952 1.72095238 3.41809524 5.1152381  6.81238095 8.50952381]
```

Affichons maintenant les points expérimentaux et la droite modélisée sur le même graphique :

**NOTE CODAGE : "r+" à la ligne 2 permet d'afficher un + rouge pour chaque point expérimental alors que "b-" à la ligne 3 permet de relier les points modélisés par des segments de droite bleus.**

[5]:
```
fig = plt.figure(figsize=(12,10))
plt.plot(U,I,'r+',label='U=f(I)')
plt.plot(Umodel,I,'b-',label='modèle linéaire')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension d'un dipôle ohmique")
plt.show()
```




**Voici le programme dans sa totalité afin d'y voir un peu plus clair !**

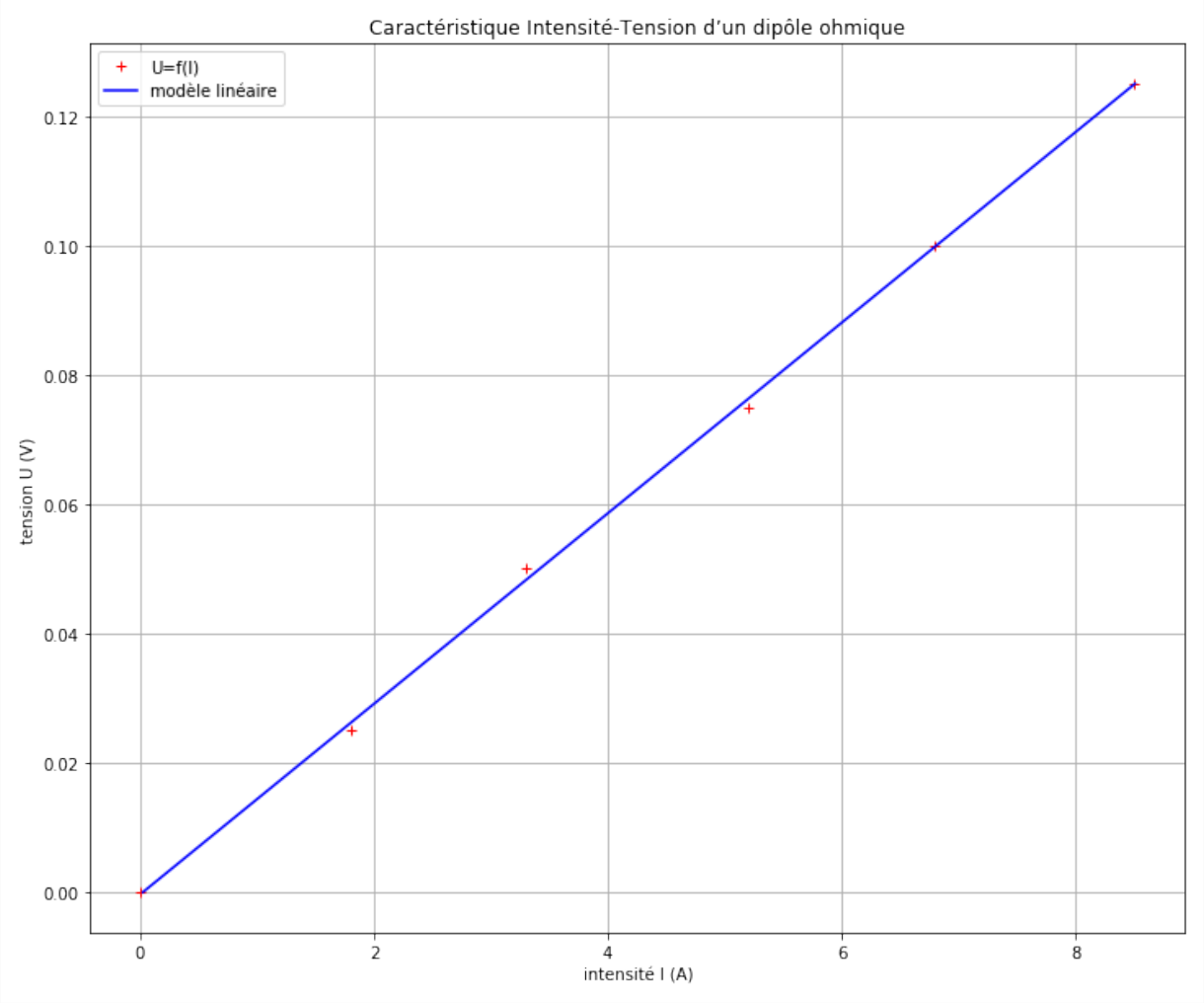
[6]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
U=np.array([0,1.8,3.3,5.2,6.8,8.5])

coeff=np.polyfit(I, U,1)
Umodel = coeff[0]*I+coeff[1]
print('U={0:.1f}'.format(coeff[0]),'xI+{0:.1f}'.format(coeff[1]))

fig = plt.figure(figsize=(12,10))
plt.plot(U,I,'r+',label='U=f(I)')
plt.plot(Umodel,I,'b-',label='modèle linéaire')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension d'un dipôle ohmique")
plt.show()
```

```
U=67.9 xI+0.0
```




#### 2.1.12.2 Deuxième partie : Modélisation à l'aide de la fonction linregress du module stats de la bibliothèque scipy

Afin de réaliser une régression linéaire, il est aussi possible d'utiliser la fonction **linregress** issue du module stats de la bibliothèque scipy :

**slope, intercept, r_value, p_value, std_error = stats.linregress(x,y)**.

Cette fonction est un plus ardue d'utilisation pour des débutants mais permet d'obtenir la valeur du coefficient de corrélation. Elle retourne cinq variables dont seules les 3 premières sont utiles en spécialité physique : - slope : le coefficient directeur (ou pente) - intercept : l'ordonnée à l'origine - r_value : le coefficient de correlation - p_value : « valeur de p bilatérale pour un test d'hypothèse dont l'hypothèse nulle est que la pente est nulle » : cette valeur sera d'autant plus grande que la probabilité d'avoir une pente nulle est importante. Par contre, si la probabilité d'avoir une pente nulle est nulle, alors cette valeur sera nulle elle-aussi. - std_error : « Erreur standard d'estimation » : plus cette valeur est faible et plus l'estimation de la pente est précise.

Pour plus d'informations : https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html

Modélisons maintenant la courbe obtenue par une droite et affichons les variables de sortie qui nous intéresse :

[7]:
```
from scipy import stats
slope, intercept, r_value, p_value, std_error = stats.linregress(I,U)
print ('pente :',slope)
print ("ordonnée à l'origine :", intercept)
print ('coefficient de corrélation r :',r_value)
```

```
pente : 67.88571428571429
ordonnée à l'origine : 0.023809523809523725
coefficient de corrélation r : 0.999720478354276
```

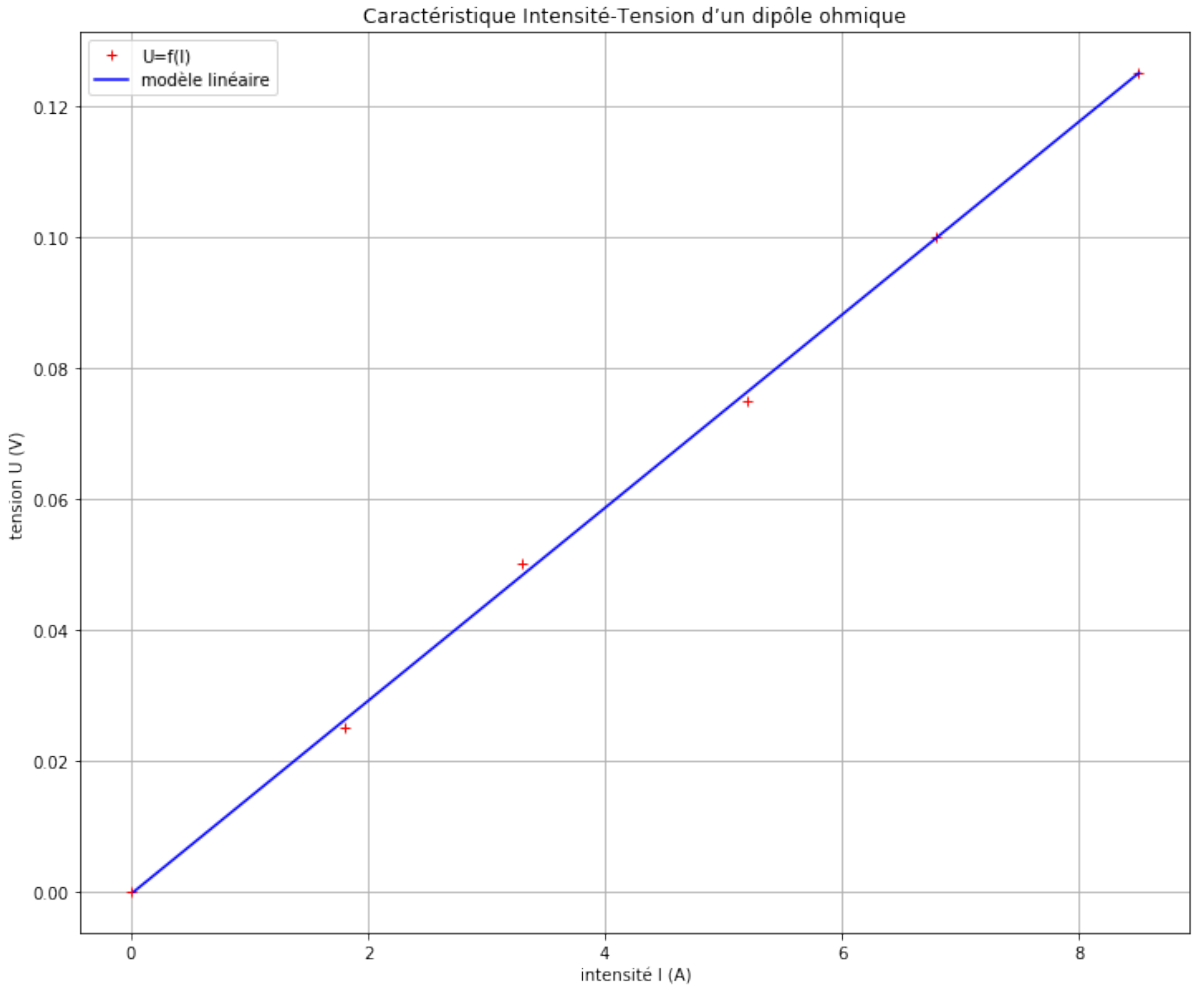
Créons l'équation de la droite modélisée et affichons cette équation ainsi que son tableau de valeurs :

[8]:
```
Umodel = slope*I+intercept
print ('U={0:.1f}'.format(slope),'xI+{0:.1f}'.format(intercept))
print(Umodel)
```

```
U=67.9 xI+0.0
[0.02380952 1.72095238 3.41809524 5.1152381  6.81238095 8.50952381]
```

Affichons maintenant les points expérimentaux et la droite modélisée sur le même graphique :

[9]:
```
fig = plt.figure(figsize=(12,10))
plt.plot(U,I,'r+',label='U=f(I)')
plt.plot(Umodel,I,'b-',label='modèle linéaire')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension d'un dipôle ohmique")
plt.show()
```




**Voici le programme dans sa totalité afin d'y voir un peu plus clair !**

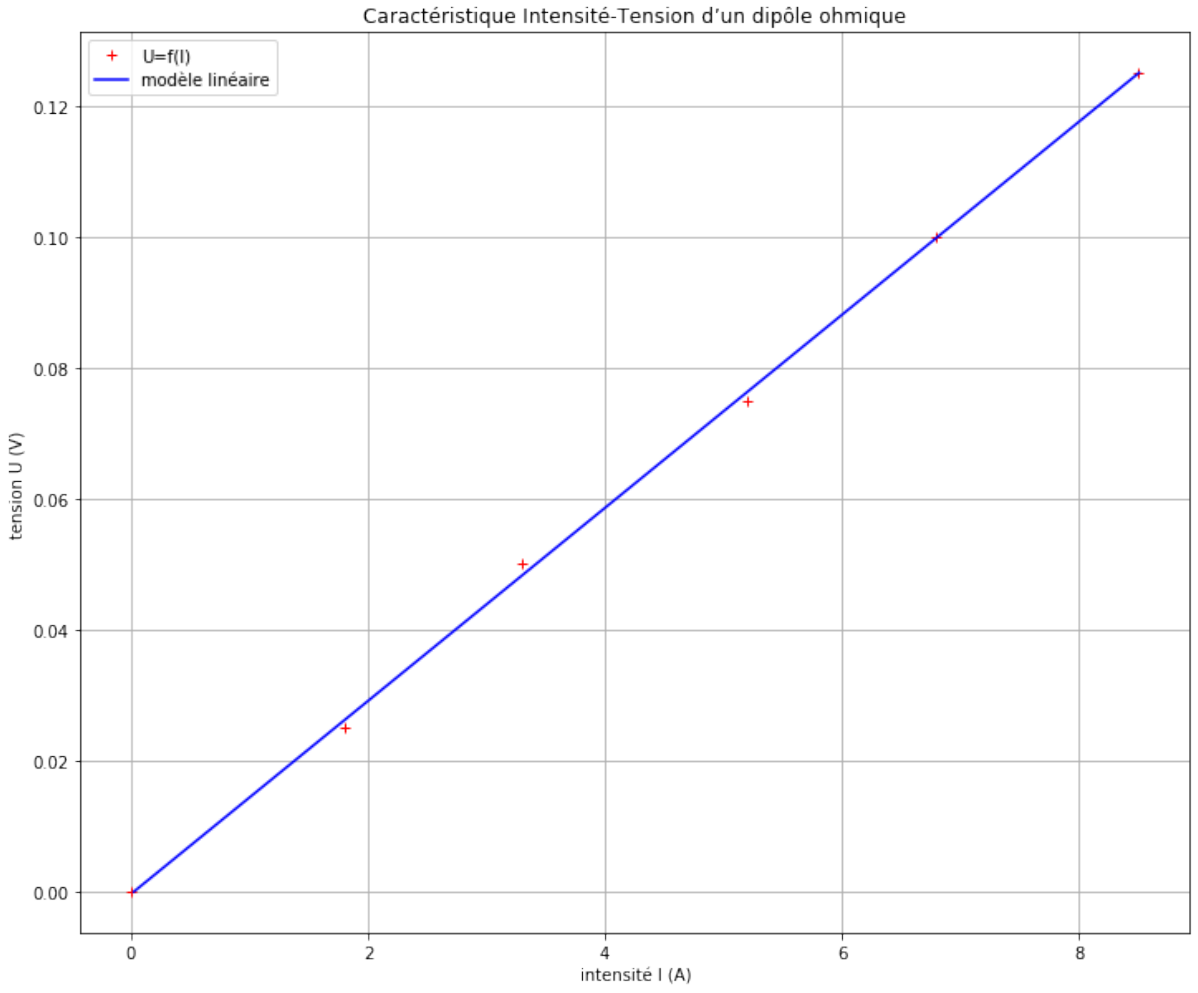
[10]:
```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
%matplotlib inline

I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
U=np.array([0,1.8,3.3,5.2,6.8,8.5])

slope, intercept, r_value, p_value, std_error = stats.linregress(I,U)
Umodel = slope*I+intercept
print ('U={0:.1f}'.format(slope),'xI+{0:.1f}'.format(intercept))
print ('coefficient de corrélation r :',r_value)


fig = plt.figure(figsize=(12,10))
plt.plot(U,I,'r+',label='U=f(I)')
plt.plot(Umodel,I,'b-',label='modèle linéaire')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension d'un dipôle ohmique")
plt.show()
```

```
U=67.9 xI+0.0
coefficient de corrélation r : 0.999720478354276
```



### 2.1.13 Les animations de graphiques



Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

Une animation est une succession d'images s'affichant à intervalle de temps régulier. Le principe de ce programme est d'afficher les courbes les unes après les autres afin de créer une courbe animée.

Pour réaliser une courbe animée, il faut utiliser la méthode :

**animation.FuncAnimation(fig, animate, init_func=init, frames=100, interval=30, blit=True,repeat=False)**

Les paramètres sont les suivants :

- fig : nom de la fenêtre graphique contenant la courbe à animer
- animate : nom de la fonction permettant de tracer chaque image sucessivement à l'aide du paramètre i.
- init_func : paramètre appelant la fonction init permettant de tracer le fond de chaque image à partir de l'objet fig
- frames : nombre d'images constituant l'animation (ici 100 images)
- interval : intervalle de temps séparant deux images successives en milliseconde (ici 30 ms)
- blit : seuls les éléments de l'image modifiés seront redessinées à chaque image si blit est égal à True
- repeat : l'animation ne s'exécutera qu'une fois, ne se repètera pas si repeat est égal à False.

Il faut donc créer le graphique et les deux fonctions init et animate avant d'utiliser la méthode **animation.FuncAnimation()**

Pour afficher l'animation dans un éditeur Python, la méthode **plt.show()** suffit.

Pour l'afficher dans ce notebook, on est obligé de passer par un widget Javascript interactif (représentation HTML par défaut des objets Animation) en utilisant les lignes de code

- **rc("animation", html="jshtml")**
- **anim** (nom de l'animation dans le programme suivant)

Afin d'illustrer ce tutoriel, nous prendrons l'exemple de l'animation d'une onde progressive sinusoïdale.

Voici le programme complet expliqué point par point :

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import animation, rc

Ymax=0.2    # amplitude en m
T=1         # période en s
l=0.2       # longueur d'onde en m

xmin=0     # valeur minimale de x
xmax=3*l   # valeur maximale de x afin d'afficher
           # 3 longueurs d'onde à l'écran
nbx=100    # nombre de points sur le graphique

# création du graphique (cf. tutoriels sur les graphiques)

fig = plt.figure(figsize=(12,10))
plt.xlim(xmin,xmax)
plt.ylim(-Ymax,Ymax)
plt.grid()
plt.xlabel("x(m)")
plt.ylabel("y(m)")
plt.title("animation : propagation d'une onde progressive "
          "sinusoïdale")

# création de la courbe que nous voulons animer.
```

(continues on next page)

(suite de la page précédente)

```
# Les listes contenant les valeurs de l'abscisse
# et de l'ordonnée sont vides.
# 'bo-' équivaut à color ='b',marker='o',styleline = '-'
# L'indice de cette courbe vaut 0.

line = plt.plot([0.3],[0],'r+',[], [],'bo-')


# création de la fonction init : permet de tracer le fond
# de chaque image à partir de la courbe définie ci-dessus
# la méthode set_data permet de remplacer les valeurs des
# coordonnées des points de la courbe (d'indice 0) de
# l'objet line par des listes vides.

def init():
    line[1].set_data([], [])
    return (line)

# création de la fonction animate : permet de tracer
# la courbe sur chaque image.
# i = indice de l'image (0 pour la première image,
# ..., n-1 pour la n ième image)

def animate(i):
    dt=0.03     # période d'échantillonnage en seconde
    t=i*dt

    # création du tableau numpy contenant les valeurs de x
    x = np.linspace(xmin,xmax,nbx)

    # écriture de la fonction définissant y en fonction
    # de x et de t.
    y = Ymax*np.cos(2 * np.pi * (x/l - t/T))

    # la méthode set_data permet de remplacer les valeurs
    # des coordonnées des points de la courbe d'indice 0
    # de l'objet line par les valeurs de x et y
    line[1].set_data(x, y)
    return (line)

# appel de la méthode animation.FuncAnimation() qui
# permettra d'afficher successivement les images à
# intervalle de temps régulier
# le paramètre interval en ms doit être égal à la
# période d'échantillonnage dt en seconde pour que le
# temps t dans l'animation corresponde au temps "réel"

anim=animation.FuncAnimation(fig, animate, init_func=init,
                             frames=100, interval=30, blit=True,
                             repeat=False)

# fermeture des fenêtres une fois affichées afin de
# gagner de la mémoire
plt.close()

# Lignes de commandes permettant de créer un widget Javascript
# interactif (représentation HTML par défaut des objets Animation)
# à remplacer par la méthode plt.show() dans un éditeur python du
# type spyder.
# attention au paramétrage dans spyder :
# menu outils> préférences> console IPython> Graphiques>
```

(continues on next page)

(suite de la page précédente)

```
# sortie graphique : automatique > OK
# puis redémarrer spyder.
# L'animation s'affichera dans une nouvelle fenêtre au lieu de
# donner un graphique vierge dans le terminal)

rc('animation', html='jshtml')
anim

# patience, l'exécution prend du temps.
```

[1]:
```
<matplotlib.animation.FuncAnimation at 0x7fa0fdd02208>
```

[ ]:

### 2.1.14 Histogrammes, étude statistique et incertitude-type de répétabilité

**Comment tracer un histogramme, réaliser une étude statistique et calculer une incertitude-type de répétabilité (type A) sous Python ?**



`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

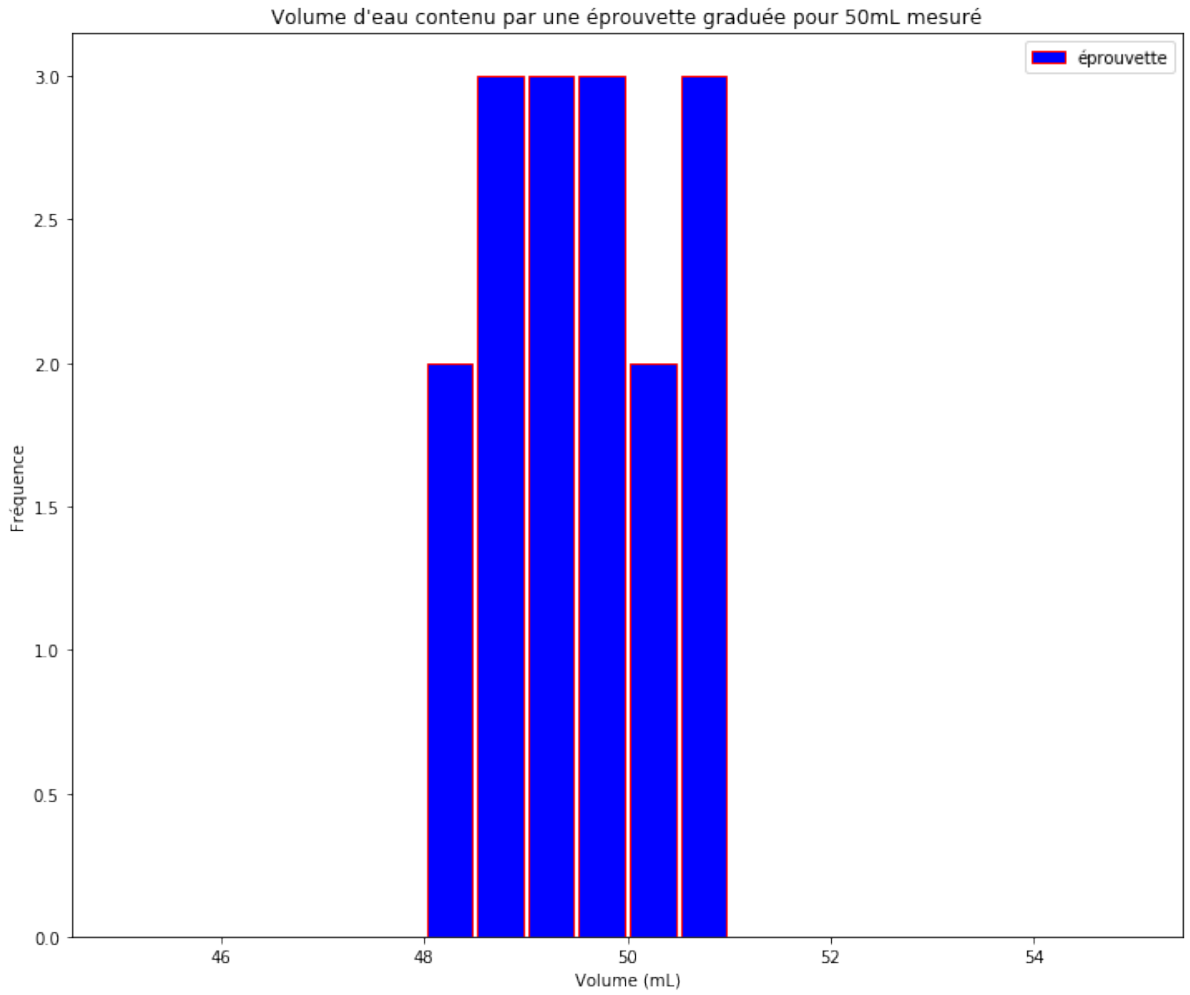
Ceci est un petit tutoriel permettant de tracer des histogrammes, de déterminer une valeur moyenne, un écart-type et une incertitude-type de répétabilité (type A) à partir d'un échantillon de valeurs.

Voici un programme exploitant des mesures de volume réalisées à l'éprouvette graduée. On mesure le plus précisément possible 50 mL d'eau à l'aide d'une éprouvette graduée. Puis, on pèse cette eau et on en déduit par un simple calcul faisant intervenir la masse volumique, le volume d'eau correspondant avec une meilleure précision. On réalise la manipulation un grand nombre de fois, ce qui nous permet d'obtenir un échantillon de valeurs.

Dans un premier temps, vous est présenté le programme complet. Les lignes de code seront explicitées dans un second temps.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
Veprouvette =[49.61,49.55,50.91,50.87,48.03,50.29,48.58,48.06,50.06,50.72,48.95,49.
↪4,49.21,49.31,49.78,48.77]
plt.figure(figsize=(12,10))
plt.hist(Veprouvette,bins=20,range=(45,55),align="mid",rwidth=0.9,color="b",
↪edgecolor="red",label="éprouvette")
plt.title("Volume d'eau contenu par une éprouvette graduée pour 50mL mesuré")
plt.xlabel("Volume (mL)")
plt.ylabel("Fréquence")
plt.legend()
plt.show()


Vmoy=np.mean(Veprouvette)
ecarttype=np.std(Veprouvette)
effectif=len(Veprouvette)
incertitudetype=ecarttype/np.sqrt(effectif)
print("{0:55}".format("Le volume moyen :"),Vmoy,"mL")
print("{0:55}".format("L'écart-type vaut :"),ecarttype,"mL")
print("{0:55}".format("Le nombre de mesures vaut :"),effectif)
print("{0:55}".format("L'incertitude-type de répétabilité de deltat vaut :"),
↪incertitudetype,"mL")
```




```
Le volume moyen :                                          49.506249999999994 mL
L'écart-type vaut :                                        0.8814042418209694 mL
Le nombre de mesures vaut :                                16
L'incertitude-type de répétabilité de deltat vaut :        0.22035106045524236 mL
```

#### 2.1.14.1 EXPLICATIONS DU PROGRAMME

Attention à bien exécuter les cellules de code suivantes les unes après les autres. Il est normal que rien ne s'affiche lors de l'exécution de certaines cellules.

#### 2.1.14.2 Import des bibliothèques utiles

[2]:
```
import matplotlib.pyplot as plt                    # bibliothèque obligatoire pour␣
↪créer des graphiques
import numpy as np                                 # bibliothèque utile pour déterminer␣
↪la valeur moyenne, l'écart-type
                                                   # et l'incertitude-type de␣
↪répétabilité
```

#### 2.1.14.3 Création de la liste contenant les valeurs de l'échantillon

Pour créer un histogramme puis réaliser l'étude statistique sous Python, il faut tout d'abord créer une liste contenant les valeurs de l'échantillon.

On peut aussi créer un tableau numpy grâce à la fonction np.array() de la bibliothèque numpy as np.

[3]:
```
Veprouvette =[49.61,49.55,50.91,50.87,48.03,50.29,48.58,48.06,50.06,50.72,48.95,49.
↪4,49.21,49.31,49.78,48.77]
```

#### 2.1.14.4 Comment créer et paramétrer un histogramme

On commence par créer une fenêtre graphique aux dimensions souhaitées grâce à la méthode **plt.figure(num, figsize, dpi)** de la bibliothèque **matplotlib.pyplot** as **plt**. Pour plus d'informations, se référer au notebook graphiques_partie_1.

Puis on crée l'histogramme à l'aide de la méthode **plt.hist(x,bins,range,align,orientation,rwidth,log,color,edgecolor,label)** de la bibliothèque **matplotlib.pyplot** as **plt**

Voici les principaux paramètres de cette méthode, sachant qu'il n'est pas obligatoire de tous les spécifier. Dans ce cas, ils prendront leur valeur par défaut. D'autres paramètres existent, pour plus d'informations : https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html?highlight=hist#matplotlib.pyplot.hist

- **x** : tableau contenant les valeurs de l'échantillon
- **bins** : le nombre d'intervalles donc de barres ; optionnel, la valeur par défaut est 10.
- **range** : donne les valeurs limites de l'axe des abscisses (xmin,xmax) ; optionnel
- **align** : {"left", "mid", "right"} ; optionnel, La valeur par défaut est "mid"

```
'left': la barre est centrée sur le côté gauche de l'intervalle
'mid': la barre est centrée sur le centre de l'intervalle
'right': la barre est centrée sur le côté droit de l'intervalle
```

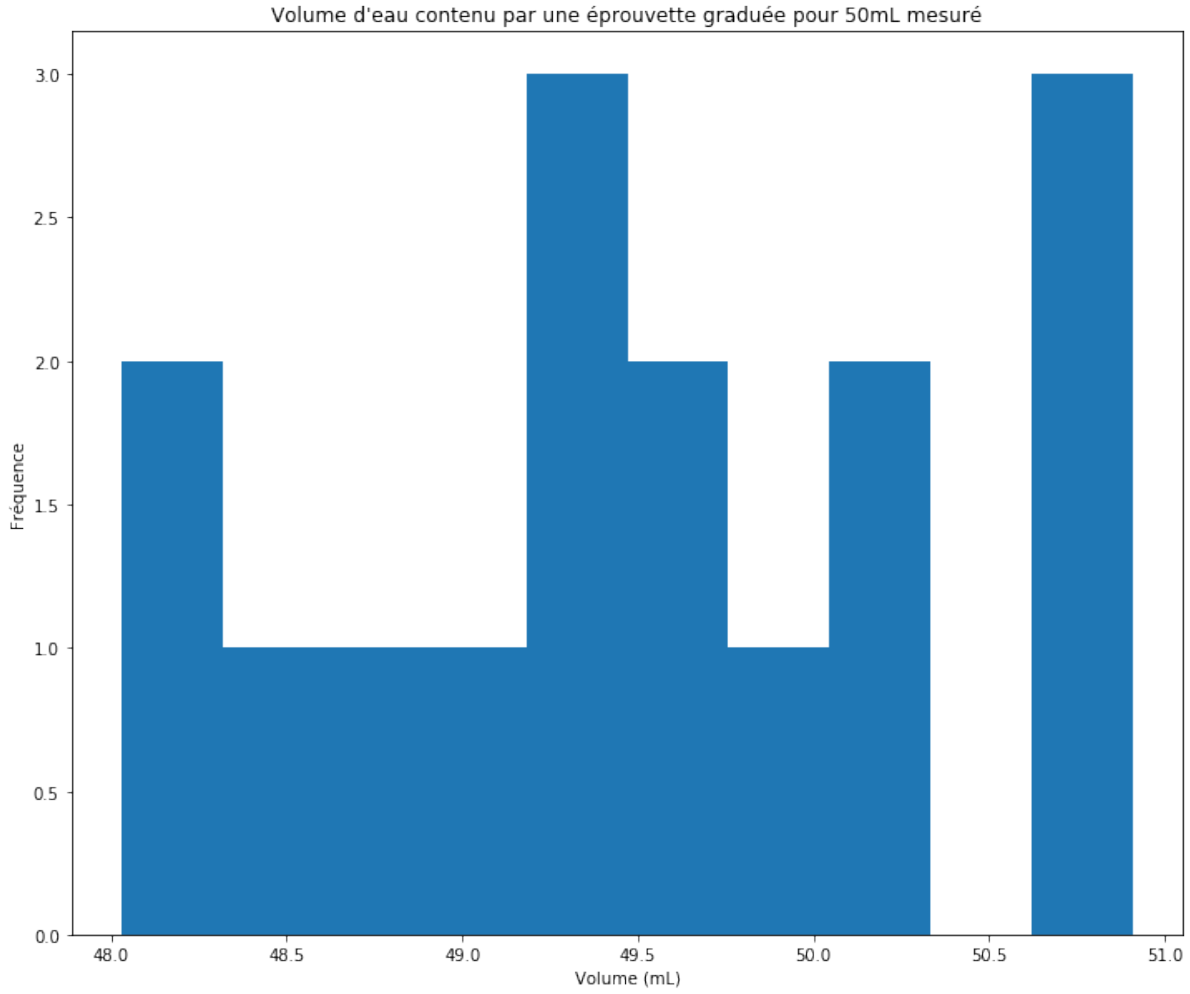
- **orientation** : {"horizontal", "vertical"} ; optionnel, la valeur par défaut est "vertical"
- **rwidth** : largeur des barres sous la forme d'une fraction de l'intervalle ; optionnel, la valeur par défaut est 1.
- **log** : booléen, permet d'afficher une échelle logarithmique pour l'axe des abscisses si log=True ; optionnel, valeur par défaut :False
- **color** : couleur des barres ; optionnel
- **edgecolor** : couleur du contour des barres ; optionnel
- **label** : permet de légender l'histogramme ; optionnel. Attention, son utilisation nécessite l'appel de la méthode **plt.legend()** dans la suite du programme

On peut ensuite afficher un titre, légender les axes de l'histogramme tout comme on le ferait sur un graphique grâce aux méthodes **plt.title(), plt.xlabel(), plt.ylabel()...** Enfin, on affiche l'histogramme grâce à la méthode **plt.show()**.

Pour plus d'informations concernant ces dernières méthodes, se référer au notebook graphiques_partie_1.

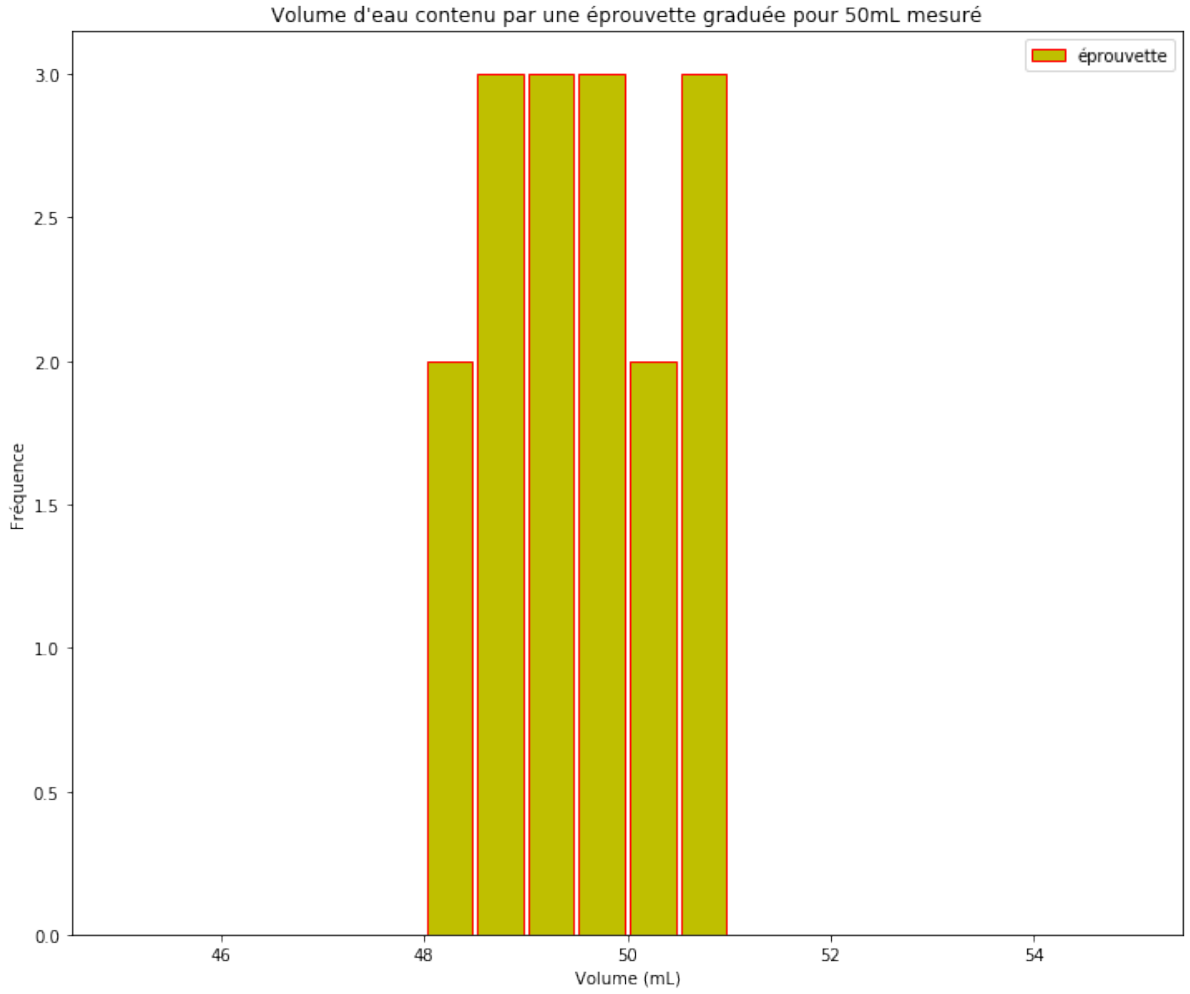
##### 2.1.14.4.1 Premier exemple : Utilisation de la méthode plt.hist() sans passage de paramètre

[4]:
```
plt.figure(figsize=(12,10))
plt.hist(Veprouvette)
plt.title("Volume d'eau contenu par une éprouvette graduée pour 50mL mesuré")
plt.xlabel("Volume (mL)")
plt.ylabel("Fréquence")
plt.show()
```




#### 2.1.14.4.2 Deuxième exemple : Utilisation de la méthode plt.hist() avec passage de paramètres

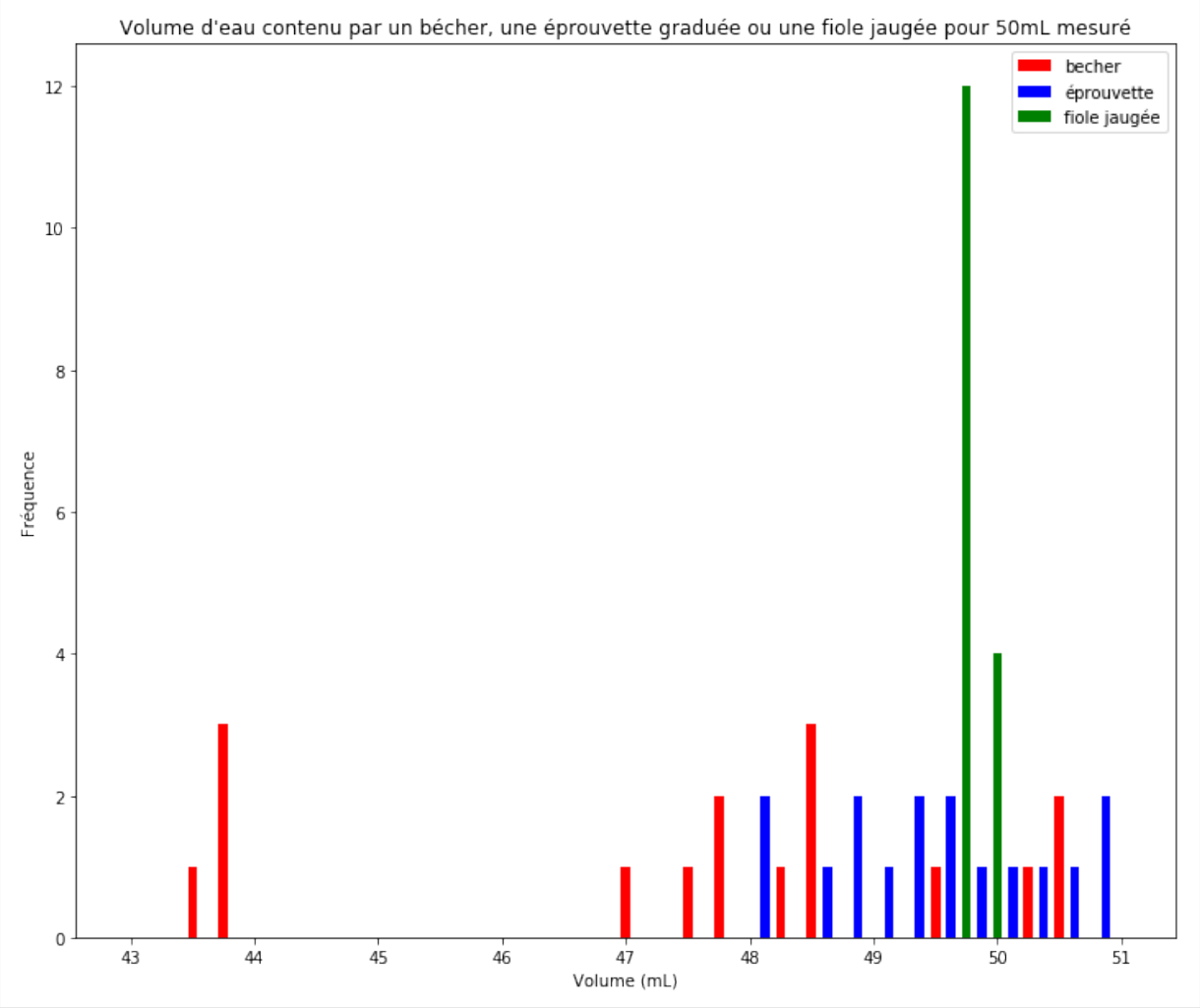
[5]:
```
plt.figure(figsize=(12,10))
plt.hist(Veprouvette,bins=20,range=(45,55),align="mid",rwidth=0.9,color="y",
↪edgecolor="r",label="éprouvette")
plt.title("Volume d'eau contenu par une éprouvette graduée pour 50mL mesuré")
plt.xlabel("Volume (mL)")
plt.ylabel("Fréquence")
plt.legend()
plt.show()
```




#### 2.1.14.4.3 Et si l'on veut afficher plusieurs histogrammes dans la même fenêtre graphique ?

Il suffit d'appeler la méthode **plt.hist()** autant de fois que nécessaire. Les paramètres sont alors très utiles !

[6]:
```
Vbecher=[50.65,48.26,47.83,47.76,50.26,47.23,43.88,43.92,48.69,48.66,43.67,47.53,
↪49.55,50.64,43.8,48.53]
Veprouvette =[49.61,49.55,50.91,50.87,48.03,50.29,48.58,48.06,50.06,50.72,48.95,49.
↪4,49.21,49.31,49.78,48.77]
Vfiole=[49.74,49.77,49.71,49.75,49.52,49.8,49.61,49.56,49.65,49.65,49.52,49.64,49.
↪74,49.81,49.5,49.59]
plt.figure(figsize=(12,10))
plt.hist(Vbecher,bins=32,range=(43,51),align="left",rwidth=0.3,color="r",label=
↪"becher")
plt.hist(Veprouvette,bins=32,range=(43,51),align="mid",rwidth=0.3,color="b",label=
↪"éprouvette")
plt.hist(Vfiole,bins=32,range=(43,51),align="right",rwidth=0.3,color="g",label=
↪"fiole jaugée")
plt.title("Volume d'eau contenu par un bécher, une éprouvette graduée ou une fiole␣
↪jaugée pour 50mL mesuré")
plt.xlabel("Volume (mL)")
plt.ylabel("Fréquence")
plt.legend()
plt.show()
```




#### 2.1.14.5 Détermination de la moyenne, de l'écart-type, de l'effectif et de l'incertitude-type de répétabilité (ou incertitude de type A)

On utilise les fonctions de la bibliothèque **numpy** as **np** :

- la fonction **np.mean(x)** permet de calculer la valeur moyenne de l'échantillon x
- la fonction **np.std(x)** permet de calculer l'écart-type de l'échantillon x
- la fonction **np.sqrt(a)** permet de calculer la racine carrée du nombre a et sera utile lors du calcul de l'incertitude-type de répétabilité.

La fonction **len(x)** permet de calculer l'effectif (nombre de valeurs) de l'échantillon x.

Pour ces trois fonctions, x peut être un tableau, un p-uplet (tuple en anglais), un tableau numpy. Pour plus de renseignements sur les listes, tuples ou les tableaux numpy, se référer aux **notebooks Listes** et **Tableaux numpy**.

L'incertitude-type de répétabilité a pour formule $U(x) = \frac{cart-type(x)}{\sqrt{effectif(x)}}$

Attention, ce programme n'arrondit pas les valeurs et n'exprime pas avec le bon nombre de chiffres significatifs la valeur moyenne et l'incertitude-type.

[7]:
```
Vmoy=np.mean(Veprouvette)
ecarttype=np.std(Veprouvette)
effectif=len(Veprouvette)
incertitudetype=ecarttype/np.sqrt(effectif)

print("{0:55}".format("Le volume moyen :"),Vmoy,"mL")
print("{0:55}".format("L'écart-type vaut :"),ecarttype,"mL")
```

(continues on next page)

(suite de la page précédente)

```
print("{0:55}".format("Le nombre de mesures vaut :"),effectif)
print("{0:55}".format("L'incertitude-type de répétabilité de deltat vaut :"),
↪incertitudetype,"mL")
```

```
Le volume moyen :                                       49.506249999999994 mL
L'écart-type vaut :                                     0.8814042418209694 mL
Le nombre de mesures vaut :                             16
L'incertitude-type de répétabilité de deltat vaut :     0.22035106045524236 mL
```

### 2.1.15 Les graphiques : zoom et pointeur

Code sous licence creative commun CC BY-NC-SA BY Gaëlle Rebolini et Jean-Matthieu Barbier

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Il est possible de zoomer et de pointer sur un graphique créé en langage Python mais uniquement dans le logiciel Jupyter Notebook (et non JupyterLab sauf après configuration du logiciel par vos soins). Pour cela, il faut importer la bibliothèque **ipywidgets as widgets** et écrire la ligne de code **%matplotlib widget** qui permet d'afficher une fenêtre interactive. Attention à ne pas ajouter la ligne de code `%matplotlib inline`.

- Placer la souris sur le point de la courbe qui vous intéresse et ses coordonnées s'affichent.
- Cliquer sur l'icône « Zoom to rectangle » et zoomer sur la courbe à l'aide la souris.
- Il est aussi possible de faire bouger les graduations des axes à l'aide de l'icône « Pan axes with left mouse, zoom with right ».
- Pour revenir à la courbe initiale, cliquer sur l'icône « Reset original view ».

[1]:
```
import matplotlib.pyplot as plt

import ipywidgets as widgets
%matplotlib widget


I=[0,25e-3,50e-3,75e-3,100e-3,125e-3]
U=[0,1.7,3.4,5.1,6.8,8.5]
plt.figure("Loi d'Ohm")
plt.plot(I,U,'b+-',label='U=f(I)')
plt.legend(loc=2)
plt.show()
```

```
Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home', 'home'),␣
↪('Back', 'Back to previous ...
```

[ ]:
```

```

## 2.2 Mémos

### 2.2.1 Glossaire

| Instructions | Bibliothèque | Rôle | Chapitre sur le site |
|---|---|---|---|
| \caractère spécial | ... | Insére un caractère spécial | Affectation des variables et impression des résultats |
| \n | ... | Retour à la ligne (chaîne de caractères) | Affectation des variables et impression des résultats |

| | | | |
|---|---|---|---|
| \t | ... | Insère une tabulation | Affectation des variables et impression des résultats |
| # | ... | Indique un commentaire | Structure d'un programme |
| % | ... | Opérateur modulo | Opérations basiques |
| ** | ... | Opérateur puissance | Opérations basiques |
| == | ... | Test d'égalité | Les boucles |
| abs | ... | Calcule la valeur absolue | Opérations basiques |
| and | ... | Opérateur logique ET | Opérations basiques |
| animation | matplotlib.pyplot | Simule le défilement d'une courbe | Comment animer un graphique? |
| append | ... | Ajoute une valeur à une liste | Les listes |
| arange | numpy | Crée un tableau | Les listes |
| array | numpy | Crée un tableau | Les listes |
| arrow | matplotlib.pyplot | Trace un vecteur | Comment créer un graphique-2? |
| axis | matplotlib.pyplot | Gère les bornes des axes | Comment créer un graphique-1? |
| break | ... | Interrompt une boucle | Les boucles |
| choice | random | Renvoie au hasard un élément d'une liste | Les listes |
| continue | ... | Saute une étape dans une boucle | Les boucles |
| def | ... | Définit une fonction | Les fonctions |
| elif | ... | Test "sinon si" | Les tests |
| else | ... | Test « sinon » | Les tests |
| end | ... | Saut à la ligne dans les affichages | Affectation des variables et impression des résultats |
| figure | matplotlib.pyplot | Crée une fenêtre graphique | Comment créer un graphique-1? |
| float | ... | Convertit vers un réel | Affectation des variables et impression des résultats |
| for...in | ... | Boucle "de .. à" | Les boucles |
| format | ... | Formate les valeurs affichées | Affectation des variables et impression des résultats |
| global | ... | Définit une variable comme étant globale | Les fonctions |
| grid | matplotlib.pyplot | Affiche une grille sur le graphique | Comment créer un graphique-1? |
| if | ... | Test « si » | Les tests |
| import | ... | Importe des bibliothèques | Structure d'un programme |
| in | ... | Vérifie si une valeur est présente | Les tests |
| input | ... | Lit ce qui est tapé sur le clavier | Affectation des variables et impression des résultats |
| insert | ... | Insère un élément dans une liste | Les listes |
| int | ... | Convertit vers un entier | Affectation des variables et impression des résultats |
| join | ... | Concatène les éléments d'une liste séparés par un séparateur | Les listes |
| legend | matplotlib.pyplot | Affiche les étiquettes d'un graphique | Comment créer un graphique-1? |
| len | ... | Renvoie la taille d'une liste | Les listes |
| list | ... | Convertit une chaîne de caractères en liste de caractères | Les listes |

| | | | |
|---|---|---|---|
| min | ... | Renvoie le plus petit élément d'une liste | Les listes |
| max | ... | Renvoie le plus grand élément d'une liste | Les listes |
| MultipleLocator | matplotlib.ticker | Gradue les axes | Comment créer un graphique-1? |
| np.linspace | numpy | Crée un tableau | Tableaux numpy |
| np.arange | numpy | Crée un tableau numpy de valeurs séquentielles | Tableaux numpy |
| np.array | numpy | Convertir une liste en tableau numpy | Tableaux numpy |
| not in | ... | Vérifie qu'une valeur ne soit pas présente | Les tests |
| open | ... | Ouvre un fichier | Comment importer des données numériques ? |
| or | ... | Opérateur logique OU | Opérations basiques |
| plot | matplotlib.pyplot | Crée un graphique | Comment créer un graphique-1? |
| polyfit | numpy | Modélise une courbe | Comment modéliser une courbe? |
| pop | ... | Supprime un élément d'une liste | Les listes |
| print | ... | Affiche à l'écran | Affectation des variables et impression des résultats |
| range | ... | Définit un intervalle | Les boucles |
| reader | ... | Lit un fichier | Comment importer des données numériques ? |
| remove | ... | Supprime un élément d'une liste | Les listes |
| return | ... | Renvoi d'une valeur par la fonction | Les fonctions |
| round | ... | Arrondit une valeur | Affectation des variables et impression des résultats |
| sample | random | Renvoie au hasard un ou plusieurs éléments d'une liste | Les listes |
| sep | ... | Insère un séparateur entre plusieurs affichages | Affectation des variables et impression des résultats |
| show | matplotlib.pyplot | Affiche le graphique | Comment créer un graphique-1? |
| sort | ... | Trie une liste | Les listes |
| sorted | ... | Trie une liste | Les listes |
| sqrt | ... | Calcule la racine carrée | Opérations basiques |
| subplot | matplotlib.pyplot | Divise la fenêtre graphique en plusieurs emplacements | Comment créer un graphique-2? |
| subplots_adjust | matplotlib.pyplot.gc | Définit la position d'un graphique dans un emplacement | Comment créer un graphique-2? |
| text | matplotlib.pyplot | Affiche un texte sur le graphique | Comment créer un graphique-1? |
| title | ... | Nomme un graphique | Comment créer un graphique-1? |
| type | ... | Indique le type d'une variable | Affectation des variables et impression des résultats |
| while | ... | Boucle « tant que » | Les boucles |
| xlabel (ou ylabel) | matplotlib.pyplot | Légende les axes | Comment créer un graphique-1? |
| xlim ( ou (ylim) | matplotlib.pyplot | Définit les bornes sur les axes | Comment créer un graphique-1? |

### 2.2.2 Syntaxe Markdown

Télécharger le pdf

Le Markdown est une manière d'écrire simplement un texte pouvant être transformé facilement en un autre type de document (pdf, html, word…) en utilisant quelques conventions pour obtenir une mise en forme ou des fonctionnalités, tout en restant facilement lisible pour un être humain.

Il est ainsi possible de créer des titres, mettre du texte en gras, en italique, créer des listes, des liens, insérer des images, des formules mathématiques, ..

```
# Un titre
## Un sous-titre

On peut formatter du texte **en gras**, en *italique*.
On peut mettre `des extraits de code`

- une liste non numérotée
- élément 2
- élément 3

Une liste numérotée :

1. liste numérotée
2. liste numérotée 2
3. liste numérotée

Pour changer de paragraphe
il faut
laisser une
une ligne blanche,
toute cette phrase ne sera que dans un paragraphe.

Prochain paragraphe

Ecrire des maths : $x^2$ ou une formule séparée : $$\frac{x^2}{\sqrt{3}}$$

```python
print("toto")
```
```

### 2.2.3 Mémo LaTeX pour les formules de physique

Télécharger le pdf

#### 2.2.3.1 Formule mathématique

Une formule mathématique se trouve entre deux caractères $. Deux modes d'affichage sont possible : en ligne, et en séparé. Une formule en ligne est affichée dans le texte : `$x^{p-2}\$` se lit $x^{2-p}$, alors qu'une formule en séparé (début et fin avec deux signes $) est affichée sur sa propre ligne : `$$\sqrt{3x-2}$$` s'affiche

$$\sqrt{3x-2}$$

#### 2.2.3.2 Notations utiles

- Exposant : `$Cl^-$` : $Cl^-$. Si plusieurs lettres en exposant : `$Mg^{2+}$` : $Mg^{2+}$
- Indice : `$CO_2$` : $CO_2$. Si plusieurs lettres en indice : `$CO_{2_{(g)}}$` : $CO_{2_{(g)}}$
- Exposant et indice `$CO_3^{2-}$` : $CO_3^{2-}$
- Espace et flèche `$2\ A+3\ B \rightarrow C$` : $2\ A + 3\ B \rightarrow C$

— Fraction : `$\frac{A}{B}$` : $\frac{A}{B}$
— Racine carrée : `$\sqrt{A}$` : $\sqrt{A}$
— Vecteur : `$\vec{u}$` : $\vec{u}$ ou `$\overrightarrow{AB}$` : $\overrightarrow{AB}$
— Somme : `$\sum_{n=1}^p n^p$` : $\sum_{n=1}^{p} n^p$

## 2.3 Activités pour la terminale

### 2.3.1 Diagramme de distribution d'un couple acido-basique

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

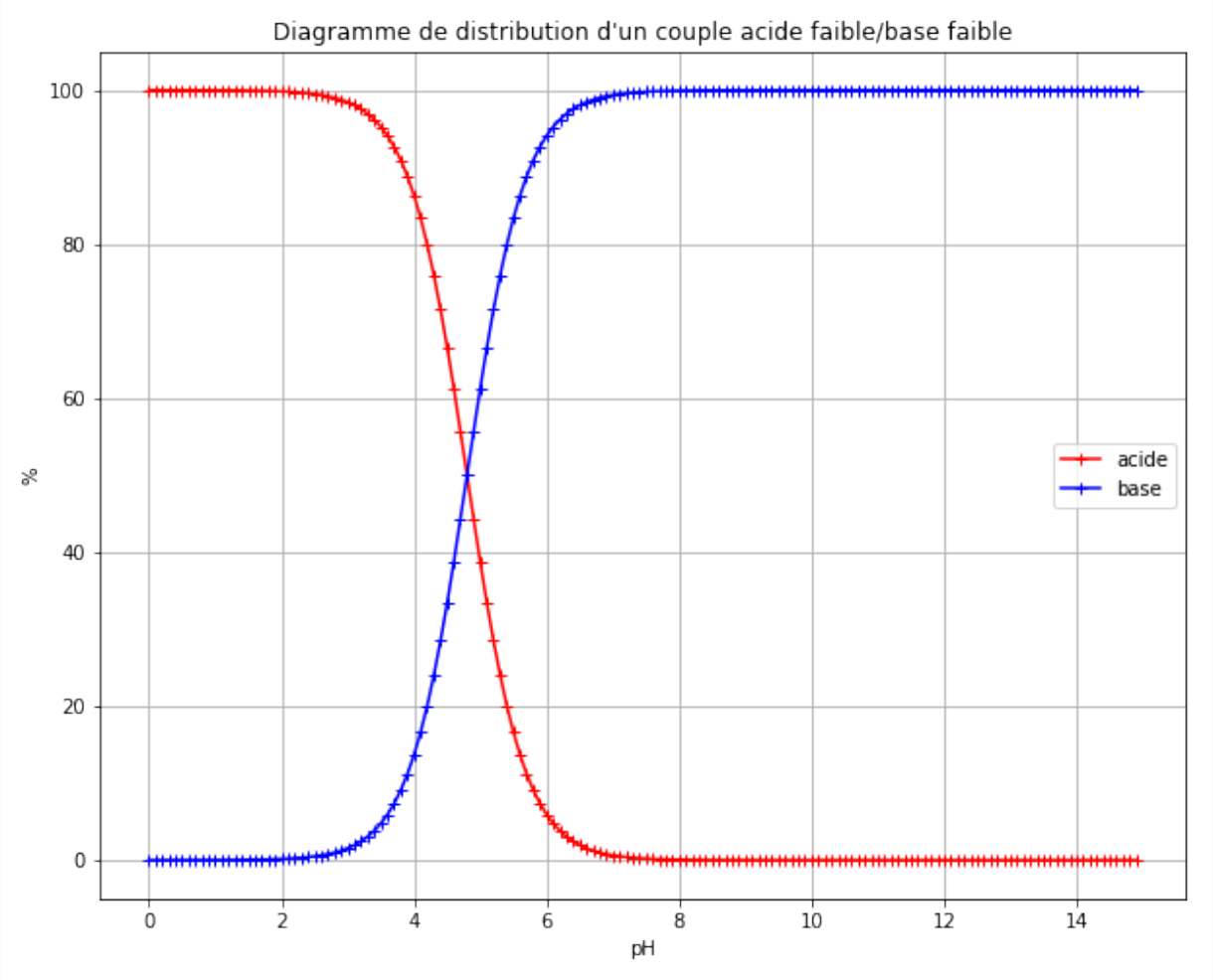
[9]:
```
# Import des bibliothèques

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

[12]:
```
def diagramme(pKa):
    pH = np.arange(0,15,0.1)
    X = 10**(pH-pKa)      # X = Cb/Ca =Pb/Pa
    Pb = X*100/(1+X)      # % de base faible
    Pa = 100-Pb           # % d'acide faible

    plt.figure (figsize = (10,8))
    plt.plot(pH,Pa,"r+-",label="acide")
    plt.plot(pH,Pb,"b+-",label="base")
    plt.xlabel("pH")
    plt.ylabel("%")
    plt.legend()
    plt.grid()
    plt.title ("Diagramme de distribution d'un couple acide faible/base faible")
    plt.show
```

[13]:
```
pKa = 4.8
diagramme (pKa)
```




[ ]:

### 2.3.2 Histogramme et évaluation de l'incertitude-type de type A sur une série de mesures

:download :'Télécharger le pdf <./histogramme.pdf>'

:download :'Télécharger le notebook <./histogramme.ipynb>'

:download :'Lancer le notebook sur binder (lent) <https ://mybinder.org/v2/gl/pyspc

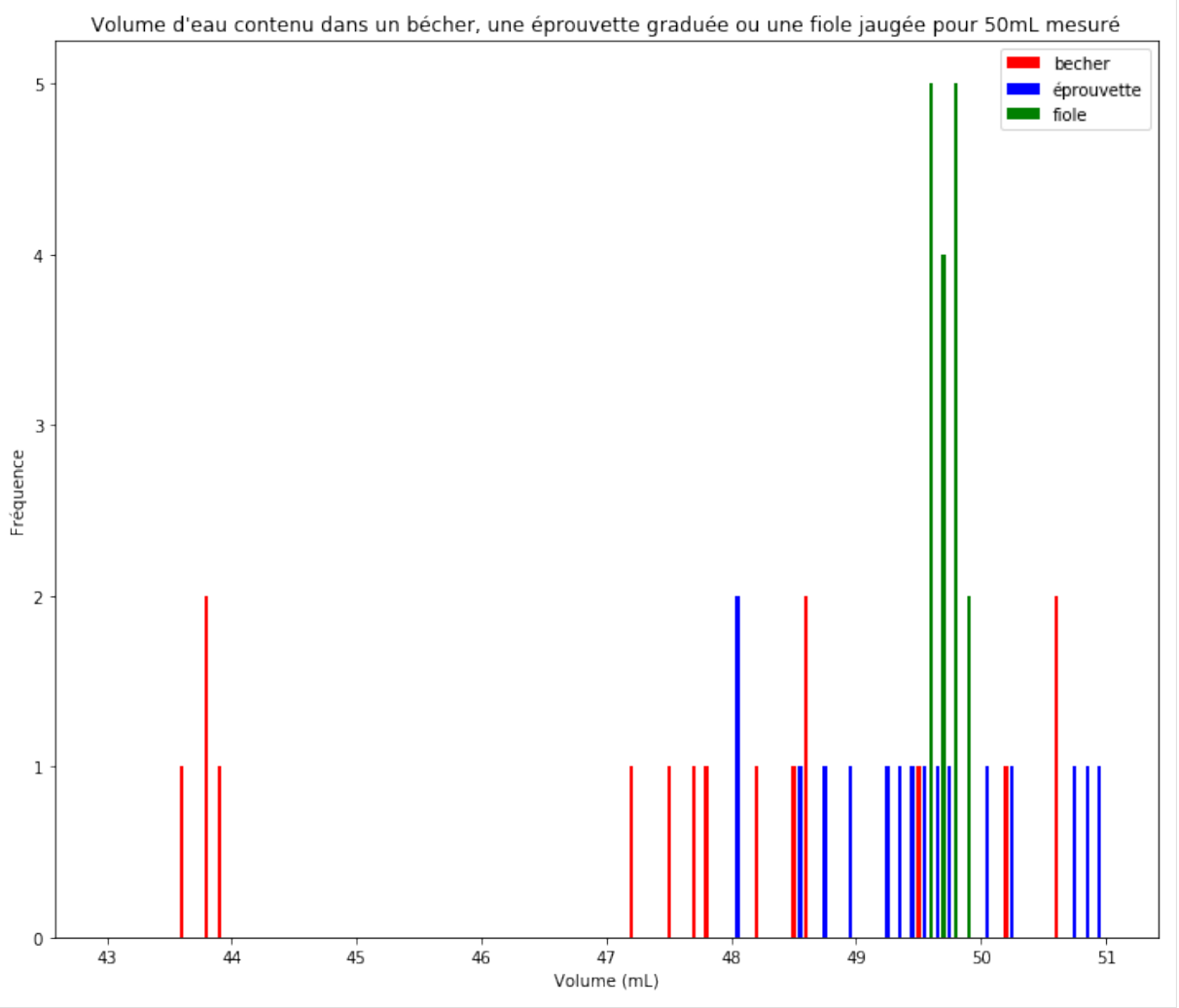
[1]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

[2]:
```
Vbecher=[50.65,48.26,47.83,47.76,50.26,47.23,43.88,43.92,48.69,48.66,43.67,47.53,
↪49.55,50.64,43.8,48.53]
Veprouvette =[49.61,49.55,50.91,50.87,48.03,50.29,48.58,48.06,50.06,50.72,48.95,49.
↪4,49.21,49.31,49.78,48.77]
Vfiole=[49.74,49.77,49.71,49.75,49.52,49.8,49.61,49.56,49.65,49.65,49.52,49.64,49.
↪74,49.81,49.5,49.59]
plt.figure(figsize=(12,10))
plt.hist(Vbecher,bins=80,range=(43,51),align="left",rwidth=0.3,color="r",label=
↪"becher")
plt.hist(Veprouvette,bins=80,range=(43,51),align="mid",rwidth=0.3,color="b",label=
↪"éprouvette")
```

(continues on next page)

(suite de la page précédente)

```
plt.hist(Vfiole,bins=80,range=(43,51),align="right",rwidth=0.3,color="g",label=
↪"fiole")
plt.title("Volume d'eau contenu dans un bécher, une éprouvette graduée ou une␣
↪fiole jaugée pour 50mL mesuré")
plt.xlabel("Volume (mL)")
plt.ylabel("Fréquence")
plt.legend()
plt.show()
```




[3]:
```
def statistique(x):
    moy=np.mean(x)
    ecarttype=np.std(x)
    effectif=len(x)
    incertitudetype=ecarttype/np.sqrt(effectif)
    return (moy,ecarttype,effectif,incertitudetype)
```

[4]:
```
statistique(Vbecher)
```

[4]:
```
(47.553749999999994, 2.378105535399974, 16, 0.5945263838499935)
```

[5]:
```
statistique(Veprouvette)
```

[5]:
```
(49.506249999999994, 0.8814042418209694, 16, 0.22035106045524236)
```

[6]:
```
statistique(Vfiole)
```

[6]:
```
(49.66, 0.10012492197250353, 16, 0.025031230493125882)
```

[ ]:
```

```

### 2.3.3 Interférences et images

Quelques exemples de l'utilisation de la bibliothèque PIL pour l'exploitation d'images. A prendre comme une piste de réflexion.

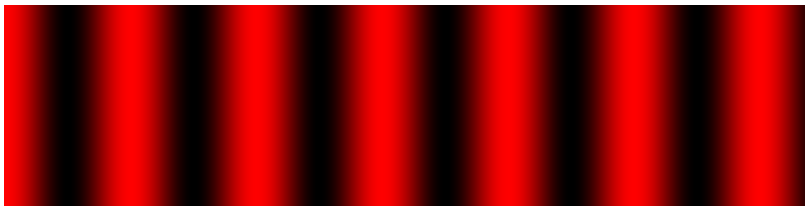
`Télécharger le pdf`

`Télécharger le notebook`, et les images associées : `diffraction` et `interferences`

`Lancer le notebook sur binder (lent)`

[2]:
```
from PIL import Image
from math import cos
def monochromatique():
    img = Image.new("RGB",(400, 100))
    for j in range(100):
        for i in range(400):
            img.putpixel((i, j), (int(255*cos(i/20)**2), 0,0))
    return img

monochromatique()
```

[2]:


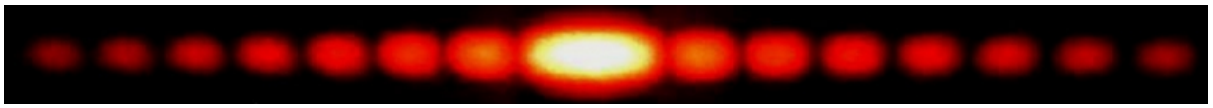
[3]:
```
import matplotlib.pyplot as plt
import numpy as np

NBVALS = 200

def get_values(image, nb):
    h = image.height
    w = image.width
    out = list()
    for i in range(nb):
        pixel = image.getpixel((
            int(i*w/nb),
            int(h/2)
        ))
        out.append(pixel[0]+pixel[1]+pixel[2])
    return out
```

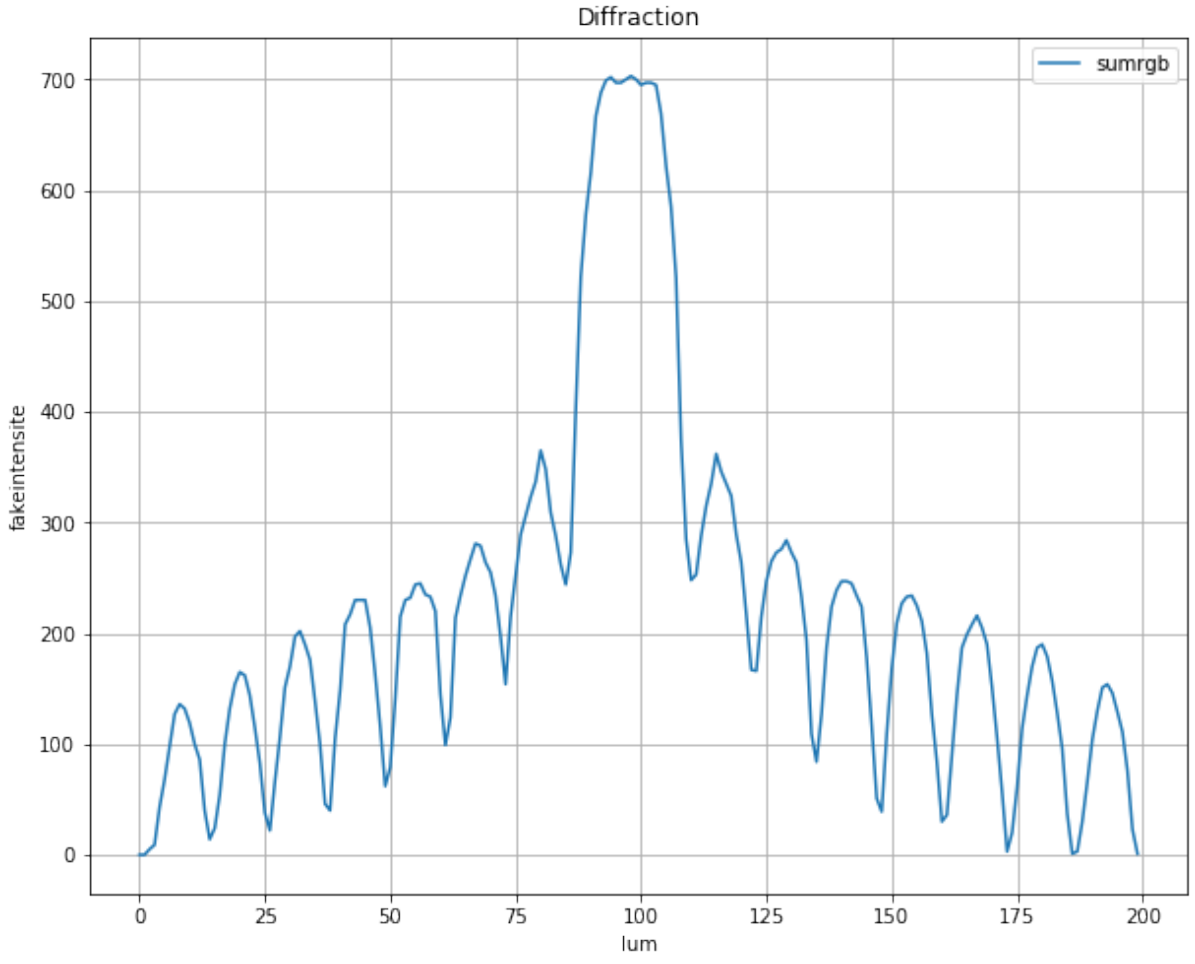
[4]:
```
# L'image utilisée ici n'est vraiment pas idéale (png d'illustration récupéré
# en ligne - saturation - image censée être monochromatique...)
# bref : à revoir avec une meilleure image - mas pas eu le temps de faire la manip␣
↪sorry
img = Image.open("./images/diffraction.png")
img
```

[4]:



[5]:

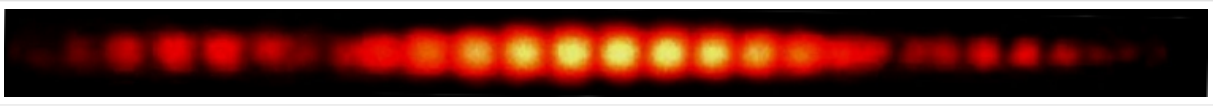
```
values = get_values(img, NBVALS)
plt.figure (figsize = (10,8))
plt.plot(range(NBVALS), values,label="sumrgb")
plt.xlabel("lum")
plt.ylabel("fakeintensite")
plt.legend()
plt.grid()
plt.title ("Diffraction")
plt.show()
```




[6]:

```
# L'image ci-dessous n'est pas d'une grande qualité non plus, désolé...
img = Image.open("./images/interferences.png")
img
```

[6]:



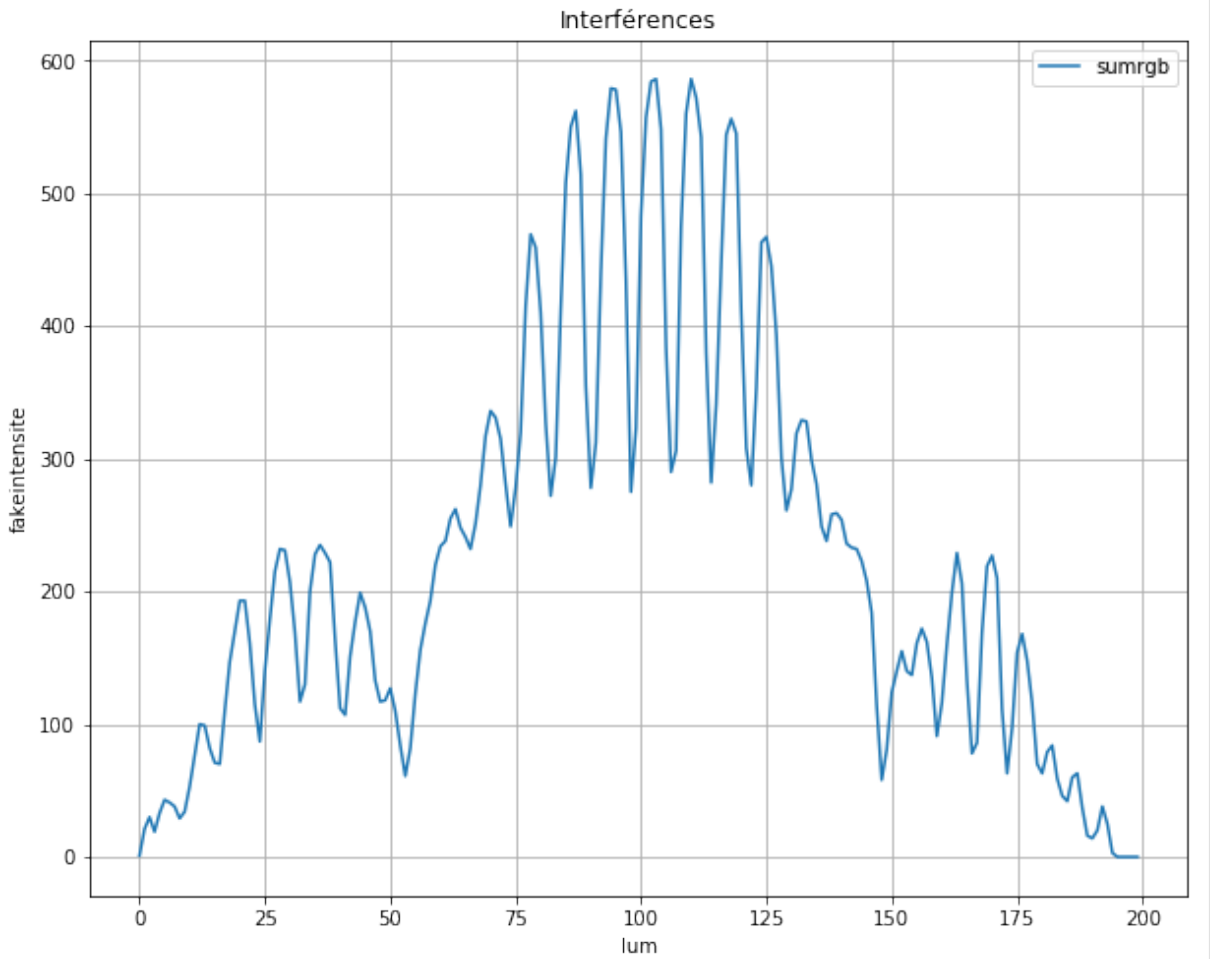
[7]:

```
values = get_values(img, NBVALS)
plt.figure (figsize = (10,8))
```

(continues on next page)

(suite de la page précédente)

```
plt.plot(range(NBVALS), values,label="sumrgb")
plt.xlabel("lum")
plt.ylabel("fakeintensite")
plt.legend()
plt.grid()
plt.title ("Interférences")
plt.show()
```




[40]:
```
# A RETRAVAILLER :)
```

[ ]:
```
```

### 2.3.4 Interférences

#### 2.3.4.1 Somme de deux signaux sinusoïdaux synchrones

Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

[1]:
```
# Import des bibliothèques

import numpy as np
```

(continues on next page)

(suite de la page précédente)

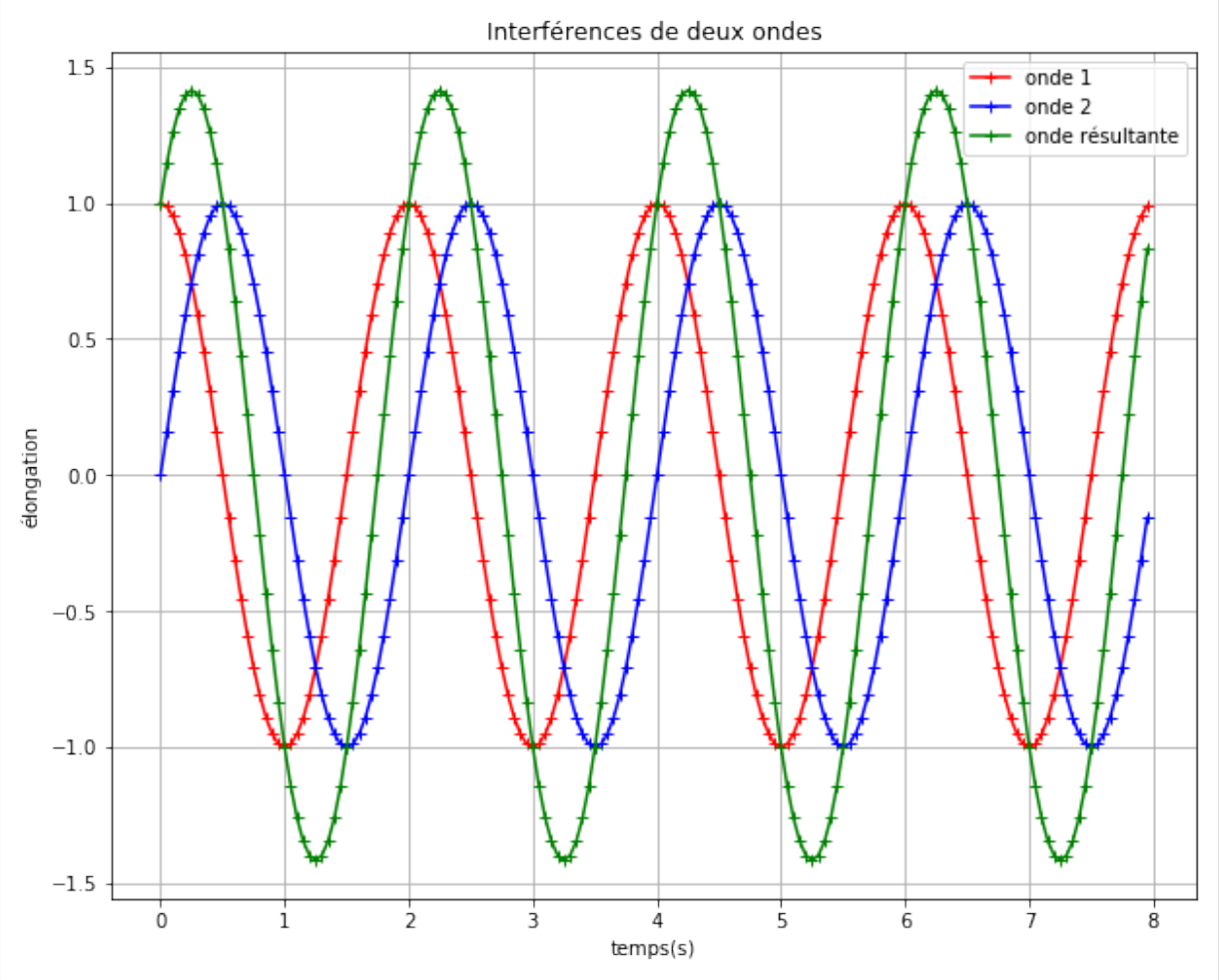
```
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
def onderesultante(A1, A2, T, phi2,duree,Te):
    t=np.arange(0,duree,Te)
    s1= A1*np.cos(2*np.pi*t/T)       # on pose phi1=0
    s2= A2*np.cos(2*np.pi*t/T+phi2)
    s=s1+s2

    plt.figure (figsize = (10,8))
    plt.plot(t,s1,"r+-",label="onde 1")
    plt.plot(t,s2,"b+-",label="onde 2")
    plt.plot(t,s,"g+-",label="onde résultante")
    plt.xlabel("temps(s)")
    plt.ylabel("élongation")
    plt.legend()
    plt.grid()
    plt.title ("Interférences de deux ondes")
    plt.show
```

[3]:
```
A1=1
A2=1
T=2   # période en s
phi2 = 1.5*np.pi
duree= 8  # durée en s
Te= 0.05

onderesultante(A1,A2,T,phi2,duree,Te)
```




### 2.3.5 Mouvement parabolique et accélération (version sans fonction)

Télécharger le pdf

Télécharger le notebook, le csv et la vidéo

Lancer le notebook sur binder (lent)

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import csv
```

[7]:
```
with open("parabole2.csv", "r", encoding="utf-8") as f:
    rparabole = csv.reader(f, delimiter=";")
    tableau=[]
    index_row=0
    N=1
    for row in rparabole:
        if index_row < N:
            index_row = index_row+1
        else :
            for i in range (len(row)):
              if len(tableau) <= i:
                  X = []
                  tableau.append(X)
```

(continues on next page)

(suite de la page précédente)

```
                try:
                    tableau[i].append(float(row[i].replace(",",'.')))
                except ValueError:
                    print('erreur:contenu de cellule non numérique')
                    continue
    print(tableau)
    t=tableau[0]
    x=tableau[1]
    y=tableau[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, 0.96, 1.0, 1.04, 1.08, 1.12, 1.16], [-0.223951998822, -0.
↪343580601208, -0.51009038223, -0.707225408616, -0.904090917691, -1.06952262947,␣
↪-1.26584910392, -1.46177130241, -1.61175563285, -1.78383296605, -1.97894661261,␣
↪-2.17352122454, -2.35318748976, -2.53150616842, -2.71835708968, -2.8809588697,␣
↪-3.06740551499, -3.24478088307, -3.42283004442, -3.60792910316, -3.78490019527,␣
↪-3.97731866872, -4.13111868882, -4.2990185037], [1.76458242983, 1.97872907661, 2.
↪22294809106, 2.45118719902, 2.66398567058, 2.84644213942, 3.02835933818, 3.
↪18711558233, 3.36984168622, 3.49040079194, 3.60283512689, 3.68438818904, 3.
↪79709215904, 3.83259294703, 3.91428082671, 3.93461061335, 3.99313753842, 3.
↪97459609901, 3.9946562506, 3.97597999367, 3.93427759966, 3.8923055706, 3.
↪85100762916, 3.73223687067], [-3.35881386319, -3.83926907511, -4.39437831948, -4.
↪6984222285, -4.61537628747, -4.59800458895, -4.6734998329, -4.39648990626, -4.
↪26517089855, -4.53285206253, -4.72307275216, -4.66513122103, -4.53973738179, -4.
↪51201266471, -4.41596392065, -4.41034897667, -4.48905549951, -4.47657956306, -4.
↪52581793104, -4.52774853362, -4.51639558597, -4.33379221852, -4.15861517319, -4.
↪15321687131], [5.54030774149, 5.7088992284, 5.72816977458, 5.44184327251, 4.
↪98744077131, 4.60697343994, 4.32141849032, 4.12782399047, 3.67426409878, 3.
↪02603583823, 2.55077455354, 2.30508951049, 1.88360398803, 1.52476923155, 1.
↪24757155895, 0.925891633931, 0.51693242311, 0.109564808036, -0.143354032685, -0.
↪647506495101, -0.990538125172, -1.24691830515, -1.81555270826, -2.43219952115]]
```

[8]:
```
vx=[]
for i in range (len(x)-1) :
    vxi=(x[i+1]-x[i])/(t[i+1]-t[i])
    vx.append(vxi)
```

[9]:
```
vy=[]
for i in range (len(y)-1) :
    vyi=(y[i+1]-y[i])/(t[i+1]-t[i])
    vy.append(vyi)
```

[10]:
```
ax=[]
for i in range (len(vx)-1) :
    axi=(vx[i+1]-vx[i])/(t[i+1]-t[i])
    ax.append(axi)
```

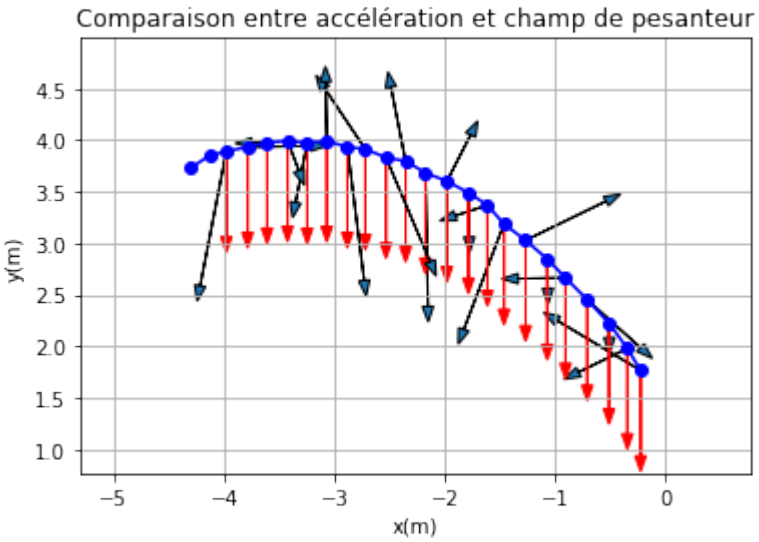
[11]:
```
ay=[]
for i in range (len(vy)-1) :
    ayi=(vy[i+1]-vy[i])/(t[i+1]-t[i])
    ay.append(ayi)
```

[12]:
```
fig = plt.figure()
plt.plot(x,y,'bo-')
for i in range (0, len (ay)):
    plt.arrow(x[i],y[i],0.03*ax[i],0.03*ay[i],head_width=0.1,
              length_includes_head=True)
    plt.arrow(x[i],y[i],0,0.1*(-9.8),fc='r',ec='r',
              head_width=0.1,length_includes_head=True)
plt.xlim(min(x)-1,max(x)+1)
```

(continues on next page)

(suite de la page précédente)

```
plt.ylim(min(y)-1,max(y)+1)
plt.grid()
plt.xlabel("x(m)")
plt.ylabel("y(m)")
plt.title("Comparaison entre accélération et "
          "champ de pesanteur")
plt.show()
```

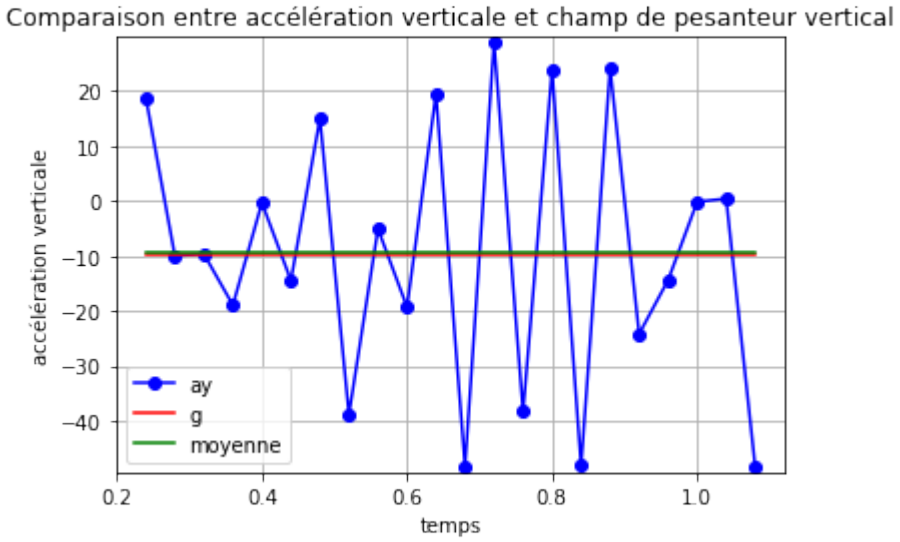



[13]:
```
t2=np.array(t[:-2])
axth=0*t2
ayth=0*t2-9.8
coeffax=np.polyfit(t2,ax,0)
axmod=0*t2+coeffax[0]
coeffay=np.polyfit(t2,ay,0)
aymod=0*t2+coeffay[0]
fig = plt.figure()
plt.plot(t2,ay,'bo-',label="ay")
plt.legend()
plt.grid()
plt.ylim(min(ay)-1,max(ay)+1)
plt.plot(t2,ayth,'r-',label="g")
plt.legend()
plt.plot(t2,aymod,'g-',label="moyenne")
plt.legend()
plt.xlabel("temps")
plt.ylabel("accélération verticale")
plt.title("Comparaison entre accélération verticale "
          "et champ de pesanteur vertical")
plt.show()
print("la valeur moyenne de l'accélération verticale est"
      ,round(coeffay[0],1),"m/s²")
plt.plot(t2,ax,'bo-',label="ax")
plt.legend()
plt.grid()
plt.ylim(min(ax)-1,max(ax)+1)
plt.plot(t2,axth,'r-',label="0")
plt.legend()
plt.plot(t2,axmod,'g-',label="moyenne")
plt.legend()
plt.xlabel("temps")
plt.ylabel("accélération horizontale")
```

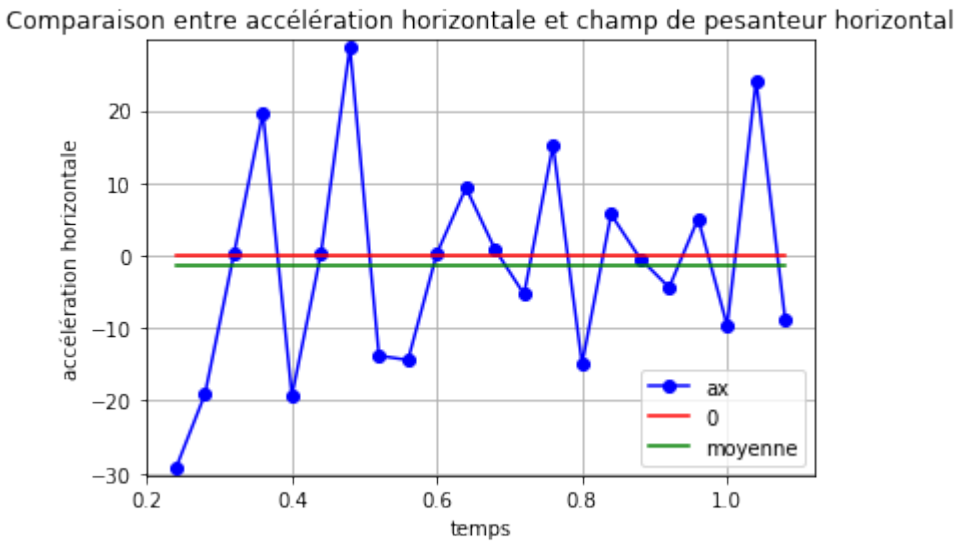
(continues on next page)

(suite de la page précédente)

```
plt.title("Comparaison entre accélération horizontale "
          "et champ de pesanteur horizontal")
plt.show()
print("la valeur moyenne de l'accélération horizontale est",
      round(coeffax[0],1),"m/s²")
```




```
la valeur moyenne de l'accélération verticale est -9.5 m/s²
```




```
la valeur moyenne de l'accélération horizontale est -1.4 m/s²
```

```
[ ]:
```

### 2.3.6 Mouvement parabolique et accélération (version avec fonctions)

Télécharger le pdf

Télécharger le notebook, le csv et la vidéo

Lancer le notebook sur binder (lent)

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import csv
```

[2]:
```
def charge_fichier_csv(fichier, delimiter=";",N=0):
    with open(fichier, 'r', encoding='utf-8') as f :
        rfichier = csv.reader(f, delimiter=delimiter)
        tableau=[]
        index_row=0
        for row in rfichier:
            if index_row < N:
                index_row = index_row+1
            else :
                for i in range (len(row)):
                    if len(tableau) <= i:
                        X = []
                        tableau.append(X)
                    try:
                        tableau[i].append(float(row[i].replace(",",'.')))
                    except ValueError:
                        print('erreur:contenu de cellule non numérique')
                        continue

        return (tableau)
```

[3]:
```
def derivee(t,z):
    dz=[]
    for i in range (len(z)-1) :
        dzi=(z[i+1]-z[i])/(t[i+1]-t[i])
        dz.append(dzi)
    return (dz)
```

[4]:
```
def graphacceleration (x,y,ax,ay):
    fig = plt.figure()
    plt.plot(x,y,'bo-')
    for i in range (0, len (ay)):
        plt.arrow(x[i],y[i],0.03*ax[i],0.03*ay[i],head_width=0.1,
                  length_includes_head=True)
        plt.arrow(x[i],y[i],0,0.1*(-9.8),fc='r',ec='r',
                  head_width=0.1,length_includes_head=True)
    plt.xlim(min(x)-1,max(x)+1)
    plt.ylim(min(y)-1,max(y)+1)
    plt.grid()
    plt.xlabel("x(m)")
    plt.ylabel("y(m)")
    plt.title("Comparaison entre accélération et champ de "
              "pesanteur")
    plt.show()
```

[5]:
```
def comparaison (t,ax,ay):
    t2=np.array(t[:-2])
    axth=0*t2
    ayth=0*t2-9.8
    coeffax=np.polyfit(t2,ax,0)
    axmod=0*t2+coeffax[0]
    coeffay=np.polyfit(t2,ay,0)
    aymod=0*t2+coeffay[0]
    fig = plt.figure()
    plt.plot(t2,ay,'bo-',label="ay")
```

(continues on next page)

(suite de la page précédente)

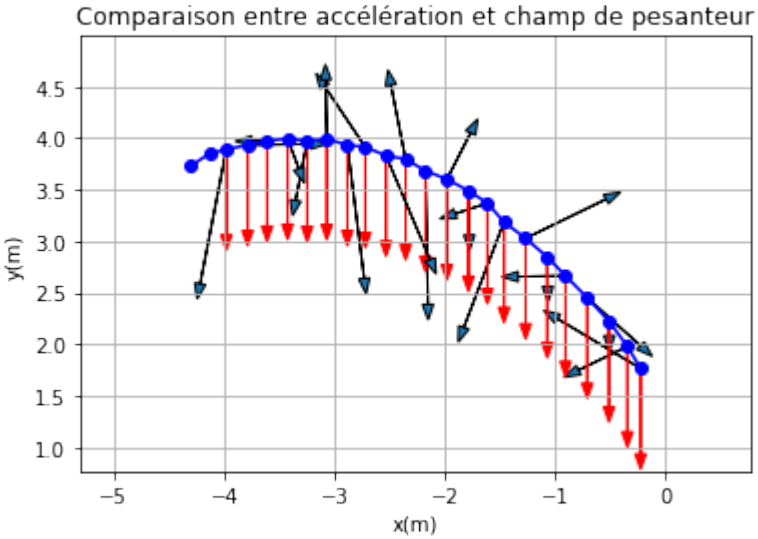
```
    plt.legend()
    plt.grid()
    plt.ylim(min(ay)-1,max(ay)+1)
    plt.plot(t2,ayth,'r-',label="g")
    plt.legend()
    plt.plot(t2,aymod,'g-',label="moyenne")
    plt.legend()
    plt.xlabel("temps")
    plt.ylabel("accélération verticale")
    plt.title("Comparaison entre accélération verticale et "
              "champ de pesanteur vertical")
    plt.show()
    print("la valeur moyenne de l'accélération verticale est"
          ,round(coeffay[0],1),"m/s²")
    plt.plot(t2,ax,'bo-',label="ax")
    plt.legend()
    plt.grid()
    plt.ylim(min(ax)-1,max(ax)+1)
    plt.plot(t2,axth,'r-',label="0")
    plt.legend()
    plt.plot(t2,axmod,'g-',label="moyenne")
    plt.legend()
    plt.xlabel("temps")
    plt.ylabel("accélération horizontale")
    plt.title("Comparaison entre accélération horizontale "
              "et champ de pesanteur horizontal")
    plt.show()
    print("la valeur moyenne de l'accélération horizontale est"
          ,round(coeffax[0],1),"m/s²")
```

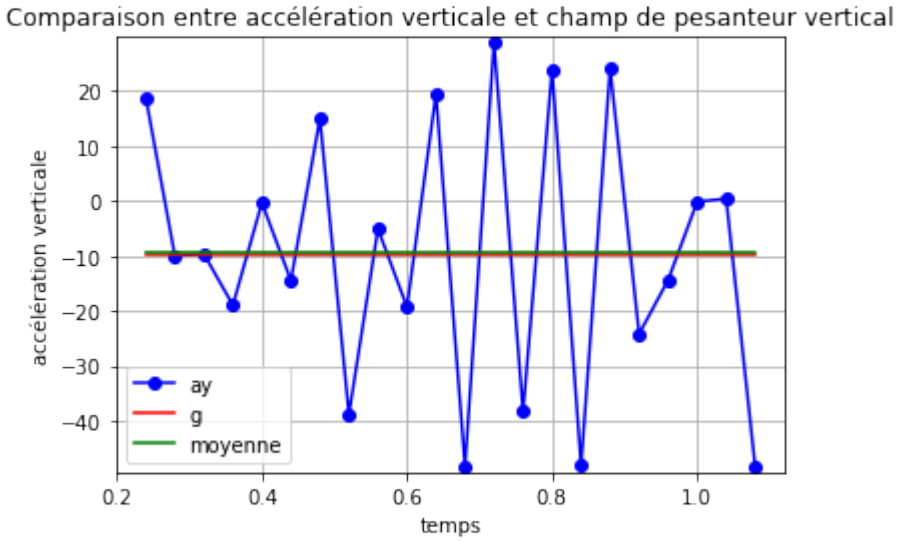
[6]:
```
tableau = charge_fichier_csv("parabole2.csv",N=1)
print(tableau)
t=tableau[0]
x=tableau[1]
y=tableau[2]
vx=derivee(t,x)
vy=derivee(t,y)
ax=derivee(t,vx)
ay=derivee(t,vy)
graphacceleration (x,y,ax,ay)
comparaison (t,ax,ay)
```

```
[[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, 0.96, 1.0, 1.04, 1.08, 1.12, 1.16], [-0.223951998822, -0.
↪343580601208, -0.51009038223, -0.707225408616, -0.904090917691, -1.06952262947,␣
↪-1.26584910392, -1.46177130241, -1.61175563285, -1.78383296605, -1.97894661261,␣
↪-2.17352122454, -2.35318748976, -2.53150616842, -2.71835708968, -2.8809588697,␣
↪-3.06740551499, -3.24478088307, -3.42283004442, -3.60792910316, -3.78490019527,␣
↪-3.97731866872, -4.13111868882, -4.2990185037], [1.76458242983, 1.97872907661, 2.
↪22294809106, 2.45118719902, 2.66398567058, 2.84644213942, 3.02835933818, 3.
↪18711558233, 3.36984168622, 3.49040079194, 3.60283512689, 3.68438818904, 3.
↪79709215904, 3.83259294703, 3.91428082671, 3.93461061335, 3.99313753842, 3.
↪97459609901, 3.9946562506, 3.97597999367, 3.93427759966, 3.8923055706, 3.
↪85100762916, 3.73223687067], [-3.35881386319, -3.83926907511, -4.39437831948, -4.
↪6984222285, -4.61537628747, -4.59800458895, -4.6734998329, -4.39648990626, -4.
↪26517089855, -4.53285206253, -4.72307275216, -4.66513122103, -4.53973738179, -4.
↪51201266471, -4.41596392065, -4.41034897667, -4.48905549951, -4.47657956306, -4.
↪52581793104, -4.52774853362, -4.51639558597, -4.33379221852, -4.15861517319, -4.
↪15321687131], [5.54030774149, 5.7088992284, 5.72816977458, 5.44184327251, 4.
↪98744077131, 4.60697343994, 4.32141849032, 4.12782399047, 3.67426409878, 3.
↪02603583823, 2.55077455354, 2.30508951049, 1.88360398803, 1.52476923155, 1.
↪24757155895, 0.925891633931, 0.51693242311, 0.109564808036, -0.143354032685, -0.
↪647506495101, -0.990538125172, -1.24691830515, -1.81555270826, -2.43219952115]]
```

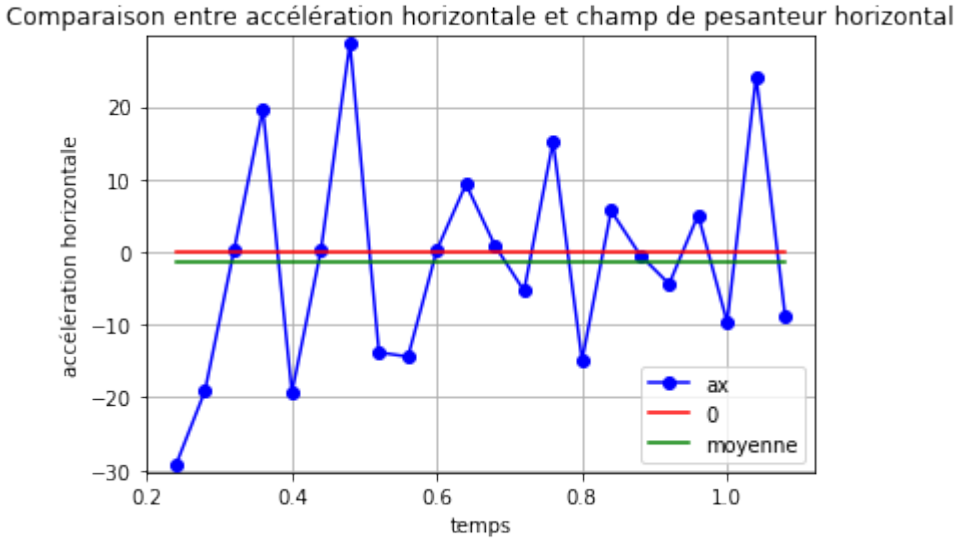
(continues on next page)

(suite de la page précédente)




```
la valeur moyenne de l'accélération verticale est -9.5 m/s²
```



```
la valeur moyenne de l'accélération horizontale est -1.4 m/s²
```

[ ]:

### 2.3.7 Réactions acido-basiques

#### 2.3.7.1 Taux d'avancement final de la réaction d'un acide faible sur l'eau

$HA_{(aq)} + H_2O_{(l)} \rightarrow A^-_{(aq)} + H_3O^+$

Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

[1]:
```
# Import des bibliothèques

import numpy as np  # ou import math
```

[2]:
```
# Première possibilité avec calcul des avancements final et maximal

def taux (Ca, pKa, V):          # solution d'acide faible de concentration Ca et␣
↪de volume V
    Ka = 10**(-pKa)
    xmax = Ca*V   # xmax = Ca*V
    # Equation du second degré : (xf/V)**2 + Ka*xf/V-Ka*Ca=0
    a = 1/(V**2)
    b = Ka/V
    c = -Ka*Ca
    delta = b**2-4*a*c
    xf1 = (-b-np.sqrt(delta))/(2*a)
    xf2 = (-b+np.sqrt(delta))/(2*a)
    if xf1<0:
        xf=xf2
    elif xf2<0:
        xf = xf1
    else :
        xf = min(xf1,xf2)
    taux = xf/xmax
    print("L'avancement final vaut ",xf," mol")
    print("L'avancement maximal vaut",xmax,"mol")
    print("et le taux d'avancement final vaut ", taux," soit ",taux*100,"%")
    return (xf,xmax,taux)
```

[3]:
```
Ca = 0.2  # concentration de la solution en acide faible apporté en mol/L
pKa = 3.8 # pKa du couple acide faible/base faible
V = 0.010 # Volume de la solution en L
taux (Ca,pKa,V)
```

```
L'avancement final vaut  5.5513986042768204e-05  mol
L'avancement maximal vaut 0.002 mol
et le taux d'avancement final vaut  0.0277569930213841  soit  2.77569930213841 %
```

[3]:
```
(5.5513986042768204e-05, 0.002, 0.0277569930213841)
```

[4]:
```
# Deuxième possibilité avec calcul de la concentration effective finale des ions␣
↪oxonium

def taux (Ca, pKa):          # solution d'acide faible de concentration Ca
```

(continues on next page)

(suite de la page précédente)

```
    Ka = 10**(-pKa)

    # Equation du second degré : Coxonium**2 + Ka*Coxonium-Ka*Ca=0
    a = 1
    b = Ka
    c = -Ka*Ca
    delta = b**2-4*a*c
    Coxonium1 = (-b-np.sqrt(delta))/(2*a)
    Coxonium2 = (-b+np.sqrt(delta))/(2*a)
    if Coxonium1<0:
        Coxonium=Coxonium2
    elif Coxonium2<0:
        Coxonium = Coxonium1
    else :
        Coxonium = min(Coxonium1,Coxonium2)
    taux = Coxonium/Ca
    print (" La concentration effective finale des ions oxonium vaut ",Coxonium,"␣
↪mol.L-1 \n"
           "et le taux d'avancement final vaut ", taux," soit ",taux*100,"%")
    return (Coxonium,taux)
```

[5]:
```
Ca = 0.2  # concentration de la solution en acide faible apporté en mol/L
pKa = 3.8 # pKa du couple acide faible/base faible
taux (Ca,pKa)
```

```
 La concentration effective finale des ions oxonium vaut  0.005551398604276821 ␣
↪mol.L-1
et le taux d'avancement final vaut  0.027756993021384103  soit  2.77569930213841 %
```

[5]:
```
(0.005551398604276821, 0.027756993021384103)
```

[ ]:
```

```

### 2.3.8 Titrage suivi par pH-métrie

#### 2.3.8.1 Titrage d'une solution aqueuse d'acide éthanoïque par une solution aqueuse d'hydroxyde de sodium

:download :'Télécharger le pdf <./titrage-ph-metrique.pdf>'

:download :'Télécharger le notebook <./titrage-ph-metrique.ipynb>'

:download :'Lancer le notebook sur binder (lent) <https ://mybinder.org/v2/gl/pyspc

[1]:
```
# Import des bibliothèques

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

[2]:
```
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])
```

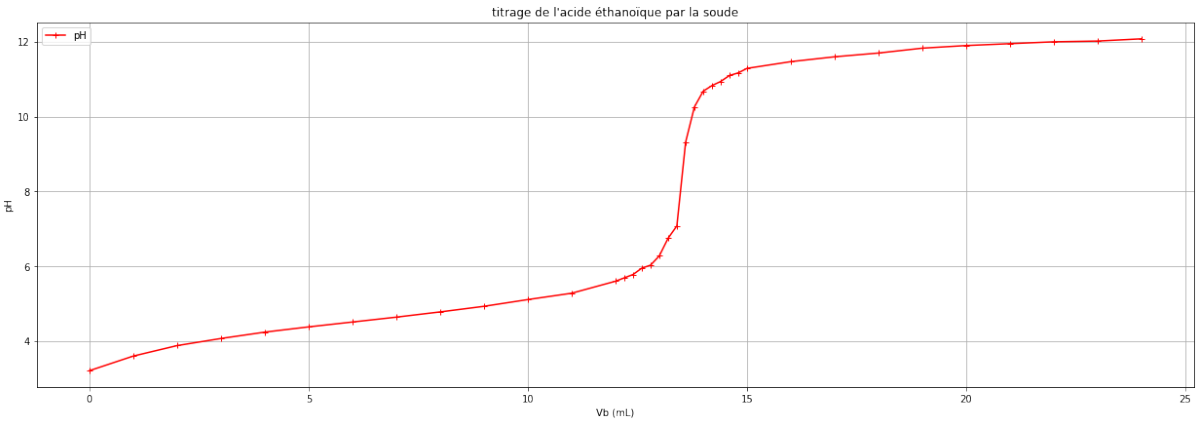
[3]:
```
def derivee(x,y):
    dery=[]
    for i in range (len(x)-1):
        deryi=(y[i+1]-y[i])/(x[i+1]-x[i])
        dery.append(deryi)
    return dery
```

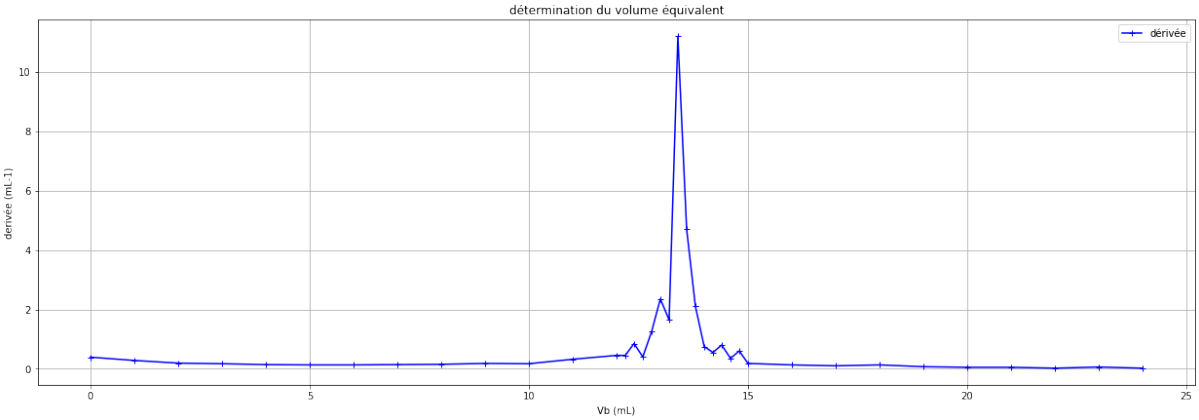
[4]:
```
derpH=derivee (Vb,pH)
print (derpH)
```

```
[0.3900000000000001, 0.2799999999999998, 0.1900000000000004, 0.16999999999999993,␣
↪0.13999999999999968, 0.1299999999999999, 0.1299999999999999, 0.14000000000000057,
↪ 0.14999999999999947, 0.1800000000000006, 0.16999999999999993, 0.
↪3199999999999994, 0.45000000000000534, 0.4499999999999969, 0.8500000000000026, 0.
↪39999999999999825, 1.2500000000000044, 2.350000000000007, 1.6499999999999915, 11.
↪20000000000004, 4.699999999999973, 2.100000000000007, 0.7500000000000044, 0.
↪5499999999999943, 0.8000000000000036, 0.34999999999999953, 0.5999999999999982, 0.
↪1800000000000015, 0.129999999999999, 0.09999999999999964, 0.13000000000000078, 0.
↪07000000000000028, 0.049999999999998934, 0.05000000000000071, 0.
↪019999999999999574, 0.0600000000000005, 0.019999999999999574]
```

[5]:
```
# Suppression de la dernière valeur du tableau à cause de l'affichage de la courbe␣
↪de la dérivée
Vb = np.delete(Vb,-1)
pH = np.delete(pH,-1)
```

[6]:
```
plt.figure(figsize=(12,10))
plt.gcf().subplots_adjust(left =0.125, bottom = 0.2, right = 1.5, top = 1.5,␣
↪wspace = 0.5, hspace = 0.5)
plt.subplot(2,1,1)
plt.plot(Vb,pH,"r+-", label="pH")
plt.xlabel("Vb (mL)")
plt.ylabel("pH")
plt.grid()
plt.title("titrage de l'acide éthanoïque par la soude")
plt.legend()
plt.subplot(2,1,2)
plt.plot(Vb,derpH,"b+-",label="dérivée")
plt.xlabel("Vb (mL)")
plt.ylabel("derivée (mL-1)")
plt.grid()
plt.title("détermination du volume équivalent")
plt.legend()
plt.show()
```




[7]:
```
# détermination du volume équivalent

Vbe = Vb[(derpH.index(max(derpH)))]
print ("Vbe=",Vbe,"mL")
```

```
Vbe= 13.4 mL
```

[8]:
```
# Evolution des quantités de matières des réactifs et prduits dans le vase␣
↪réactionnel
cb = 0.1  # concentration de la solution titrante d'hydroxyde de sodium
na=np.array([])
nb=np.array([])
nc=np.array([])
for i in range (len(Vb)):
    if Vb[i]<=Vbe:
        nai = cb*Vbe-cb*Vb[i]    # qté de matière d'acide éthanoïque en mmol
        nbi = 0                  # qté de matière des ions hydroxyde en mmol
        nci = cb*Vb[i]           # qté de matière des ions éthanoate en mmol
        na = np.append(na,nai)
        nb = np.append(nb,nbi)
        nc = np.append(nc,nci)
    else:
        nai = 0                  # qté de matière d'acide éthanoïque en mmol
        nbi = cb*(Vb[i]-Vbe)        # qté de matière des ions hydroxyde en mmol
        nci = cb*Vbe             # qté de matière des ions éthanoate en mmol
        na = np.append(na,nai)
        nb = np.append(nb,nbi)
```

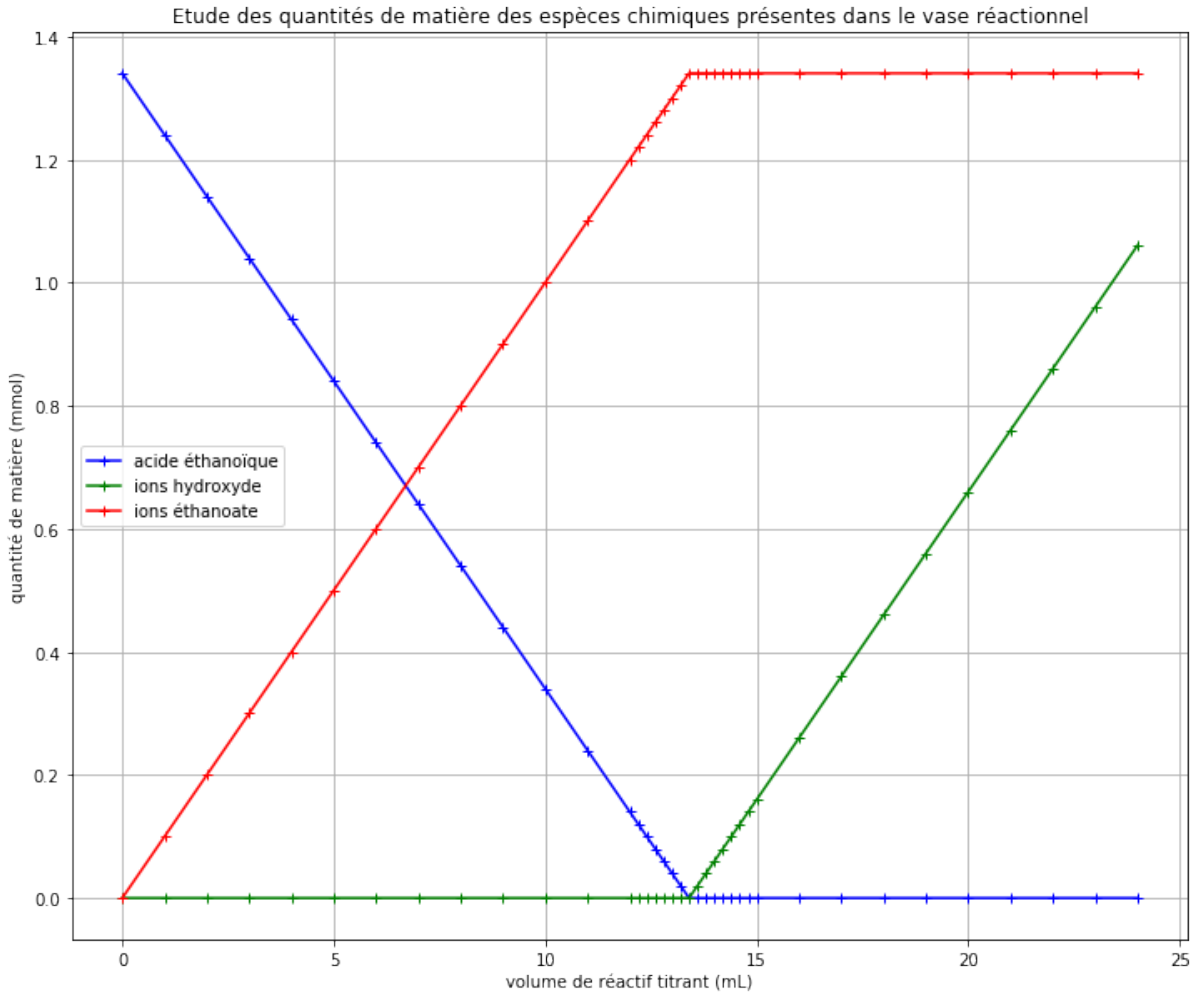
(continues on next page)

(suite de la page précédente)

```
        nc = np.append(nc,nci)
print (na)
print (nb)
print (nc)
```

```
[1.34 1.24 1.14 1.04 0.94 0.84 0.74 0.64 0.54 0.44 0.34 0.24 0.14 0.12
 0.1  0.08 0.06 0.04 0.02 0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
[0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.02 0.04 0.06 0.08 0.1  0.12 0.14 0.16
 0.26 0.36 0.46 0.56 0.66 0.76 0.86 0.96 1.06]
[0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2  1.22
 1.24 1.26 1.28 1.3  1.32 1.34 1.34 1.34 1.34 1.34 1.34 1.34 1.34 1.34
 1.34 1.34 1.34 1.34 1.34 1.34 1.34 1.34 1.34]
```

[9]:
```
plt.figure(figsize=(12,10))
plt.plot(Vb,na,"b+-",label="acide éthanoïque")
plt.plot(Vb,nb,"g+-",label="ions hydroxyde")
plt.plot(Vb,nc,"r+-",label="ions éthanoate")
plt.xlabel("volume de réactif titrant (mL)")
plt.ylabel("quantité de matière (mmol)")
plt.title("Etude des quantités de matière des espèces chimiques présentes dans le␣
↪vase réactionnel")
plt.legend()
plt.grid()
plt.show()
```



[ ]:

## 2.4 Activités pour la première

### 2.4.1 Animation d'un onde le long d'une corde

Ce notebook est long à charger, patientez :)

Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import animation, rc

# Fonction permettant l'animation d'une onde
# d'amplitude Ymax (en m), de période T (en secondes)
# et de longueur d'onde lamb (en m). L'affichage sur x
# se fait entre 0 et xmax (xmax = 3 longueurs d'onde
# si le paramètre n'est pas fourni)
def onde_corde(Ymax, T, lamb, xmax=None):
    print("Calcul de l'animation en cours, "
          "merci de patienter...")
    xmin=0
    if xmax is None:
        xmax=3*lamb
    nbx=100

    fig=plt.figure(figsize=(12,10))
    line = plt.plot([], [],'bo-')
    plt.xlim(xmin,xmax)
    plt.ylim(-Ymax,Ymax)
    plt.grid()
    plt.xlabel("x(m)")
    plt.ylabel("y(m)")
    plt.title("animation : propagation d'une "
              "onde le long d'une corde")

    def init():
        line[0].set_data([], [])
        return (line)

    def animate(i):
        dt=0.03
        t=i*dt
        x = np.linspace(xmin, xmax, nbx)
        y = Ymax*np.cos(2 * np.pi * (x/lamb - t/T))
        line[0].set_data(x, y)
        return (line)

    anim = animation.FuncAnimation(
        fig,
        animate,
        init_func=init,frames=50,
        interval=30,
        blit=True,
```

(continues on next page)

(suite de la page précédente)

```
        repeat=False)

    plt.close()

    # lignes de code à remplacer par plt.show()
    # sur un éditeur python (spyder...)
    rc('animation', html='jshtml')
    print("Calcul terminé, affichage en cours de téléchargement...")
    return anim
```

[2]:
```
# Utilisation, animation d'une onde d'amplitude
# Ymax=0,2m, de période T=1s et de longueur d'onde
# lamb=0,4m, pour un affichage jusqu'à xmax=2m
# Il faut être patient, c'est un peu long à s'afficher
onde_corde(Ymax=0.2, T=1, lamb=0.4, xmax=2)
```

```
Calcul de l'animation en cours, merci de patienter...
Calcul terminé, affichage en cours de téléchargement...
```

[2]:
```
<matplotlib.animation.FuncAnimation at 0x7f60f0eaeef0>
```

[4]:
```
# Utilisation, animation d'une onde d'amplitude
# Ymax=0,2m, de période T=1s et de longueur d'onde
# lamb=0,9m, pour un affichage jusqu'à xmax=2m
# Il faut être patient, c'est un peu long à s'afficher
onde_corde(Ymax=0.2, T=1, lamb=0.9, xmax=2)
```

```
Calcul de l'animation en cours, merci de patienter...
Calcul terminé, affichage en cours de téléchargement...
```

[4]:
```
<matplotlib.animation.FuncAnimation at 0x7f60f02d86d8>
```

### 2.4.2 PFD et analyse d'une force de frottement.

#### 2.4.2.1 (Enseignement de spécialité première S)

- Capacité numérique : 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.
- activité support : Etude de la chute verticale d'un système avec ou sans frottement (acquisition vidéo). Enregistrement à l'aide d'une table à coussin d'air du mouvement rectiligne d'un système.
- Dans ces conditions les élèves déterminent les vitesses en différents points de la trajectoire. (remarque : il est possible de modifier le programme pour calculer les vitesses en fonction des positions.)
- Le programme permet de représenter la courbe donnant l'évolution de la vitesse en fonction du temps. Il permet de modéliser les forces de frottement. Ici seul le cas des frottements f=k*v est modélisé.
- Il est possible de faire les autres modèles de frottements en modifiant la puissance sur la vitesse.

Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

[1]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#
#On importe les bibliothèques et les modules nécessaires.

from math import *
import numpy as np
```

(continues on next page)

(suite de la page précédente)

```
import matplotlib.pyplot as plt
%matplotlib inline


#définition d'une fonction permettant de calculer la dérivée à un instant donné
# les conditions initiales sont imposées par défaut.
def derivation(position1=0,position2=1,instant1=0,instant2=1):
    #initialisation d'une variable globale v
    global derivee
    derivee=0
    derivee=(position2-position1)/(instant2-instant1)



# programme principal
#Initialisation des données vitesse.
vitesse=np.array([0,14,23,29,34,37,39,40,41,42,43,43,43,43,43,43,43,43])

#initialisation des listes qui recevront les instants et l'accélération.
acceleration=[]
temps=[]

# repérage temporel des instants dans la liste temps[], à partir de la fréquence d
↪'aquisition et du nombre total de points.
for i in range(len(vitesse)):
    t=20*i
    temps.append(t)

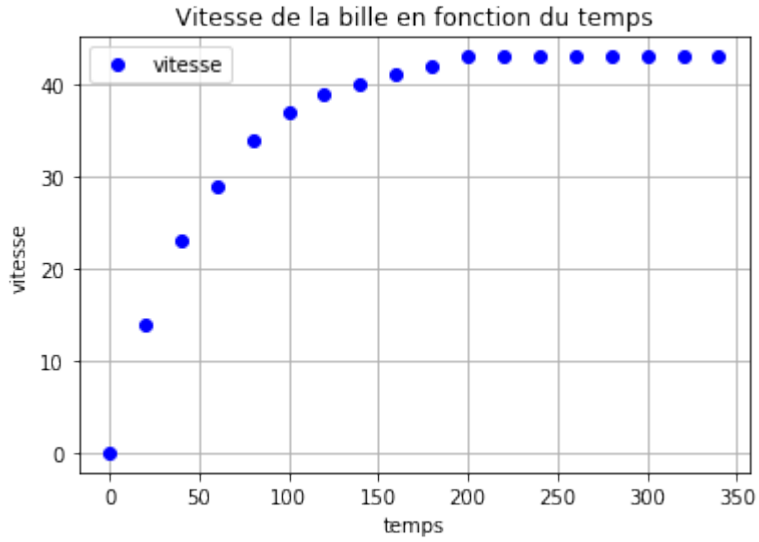
#calcul des accélération en utilisant la fonction dérivation utilisation de la␣
↪boucle for.
for i in range(len(vitesse)-1):
    derivation(vitesse[i],vitesse[i+1],temps[i],temps[i+1])
    acceleration.append(derivee)

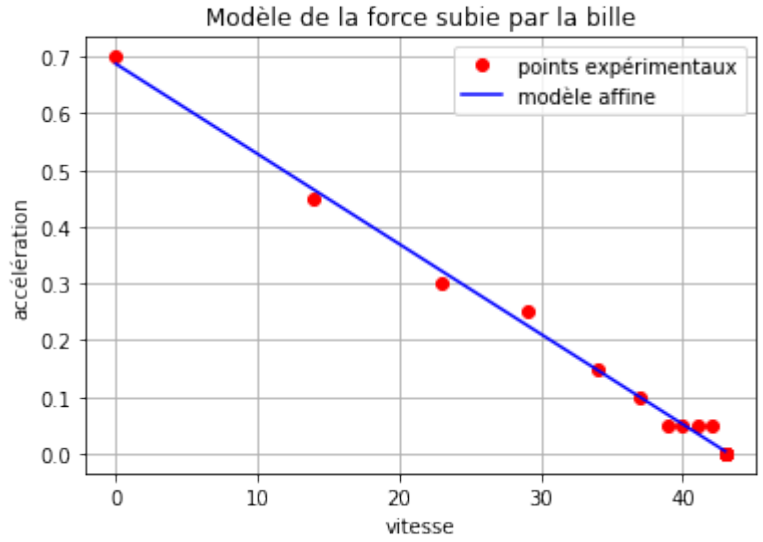

#modélisation de la force de frottement à partir de l'interprétation du PFD.
v0=vitesse[:-1]
# regression lineaire à partir de la fonction prédéfinie polyfit. Les arguments␣
↪sont la vitesse, l'accélération,
# et la puissance envisagée pour le modèle ici par défaut elle est prise égale à 1.
mod=np.polyfit(v0,acceleration,1)
model=mod[0]*v0+mod[1]
#(mod[0] coefficient directeur et mod[1] ordonnée à l'origine.
#représentation graphique pour interprétation avec le PFD.
plt.figure()
plt.plot(temps,vitesse,'bo', label='vitesse')
plt.legend()
plt.grid()
plt.xlabel("temps")
plt.ylabel("vitesse")
plt.title("Vitesse de la bille en fonction du temps")
plt.show()
#représentation de la modélisation de la force de frottement
plt.figure()
plt.plot(v0,acceleration,'ro',label='points expérimentaux')
plt.plot(v0,model,'b-',label='modèle affine')
plt.legend()
plt.grid()
plt.xlabel("vitesse")
plt.ylabel("accélération")
plt.title("Modèle de la force subie par la bille")
```

(continues on next page)

(suite de la page précédente)

```
plt.show()
```




## 2.4.3 Analyse énergétique d'un mouvement

### 2.4.3.1 (spécialité physique première S)

— Capacité numérique : Utiliser un langage de programmation pour effectuer le bilan énergétique d'un système en mouvement.

activité support :

— Les positions d'un système sont obtenues à partir d'une table à coussin d'air ou d'un enregistrement vidéo. Après avoir pour chaque point de la trajectoire précisé les coordonnées (X,Y), les élèves doivent les rentrer dans les deux listes dédiées.

— Les élèves doivent ensuite indiquer l'intervalle de temps qui sépare deux positions consécutives en modifiant la valeur par défaut 0.1 dans la ligne :t=0.1*i

— Le programme permet à partir de ces positions de calculer puis représenter les différentes énergies associées aux mouvements.Les élèves doivent pour cela modifier la valeur de la masse pour les fonctions energie_cinetique et energie_potentielle.

`Télécharger le pdf`

Télécharger le notebook

Lancer le notebook sur binder (lent)

[1]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#
#On importe les bibliothèques et les modules nécessaires.

from math import *
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


#définition d'une fonction permettant de calculer la dérivée à un instant donné
# les conditions initiales sont imposées par défaut par précaution.
def derivation(position1=0,position2=1,instant1=0,instant2=1):
    #initialisation d'une variable globale v
    global derivee
    derivee=0
    derivee=(position2-position1)/(instant2-instant1)

#définition d'une fonction premettant de calculer la norme d'un vecteur ou une␣
↪distance
def module (valeurx=0,valeury=0):
    global norme
    norme=0
    norme= sqrt(valeurx*valeurx+valeury*valeury)

#définition d'une fonction pour calculer l'énergie cinétique d'un système
# la vitesse et la masse sont définies par défaut. On peut être amener à changer␣
↪la masse.
def energie_cinetique( v=1, masse=1):
    global energiecinetique
    energiecinétique=0
    energiecinetique=0.5*masse*v*v
    #Si nécessaire pour vérifier le bon fonctionnement de la fonction retirer #␣
↪devant print.
    #print (energiecinetique)

#définition d'une fonction pour calculer l'énergie potentielle d'un système
# la position et la masse sont définies par défaut. On peut être amener à changer␣
↪la masse.
def energie_potentielle(position,masse=1):
    global energiepotentielle
    energiepotentielle=0
    energiepotentielle=masse*9.8*position
    #Si nécessaire pour vérifier le bon fonctionnement de la fonction retirer #␣
↪devant print.
    #print(energiepotentielle)

#définition d'une fonction pour calculer l'énergie potentielle d'un système
def energie_mecanique():
    global energiemecanique
    energiemecanique=0
    energiemecanique= energiepotentielle+energiecinetique
    #Si nécessaire pour vérifier le bon fonctionnement de la fonction retirer #␣
↪devant print.
    #print(energiemecanique)

def representation_graphique (abscisse=[1],ordonnee=[1],etiquette="légende",titre=
↪"Energie en fonction du temps",abs="temps(s)",ordo="Energie(J)"):
```

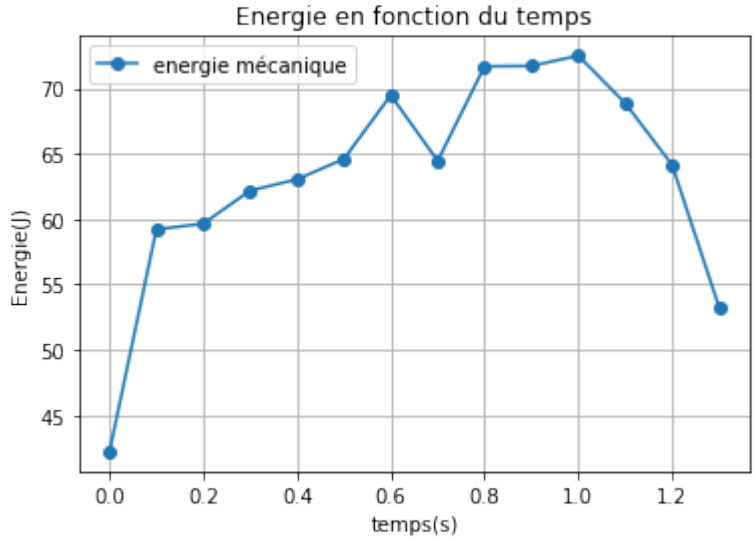
(continues on next page)

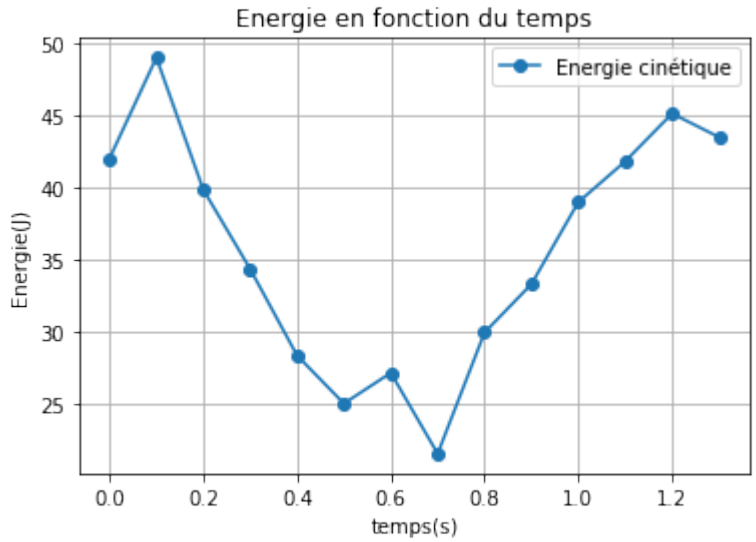
(suite de la page précédente)

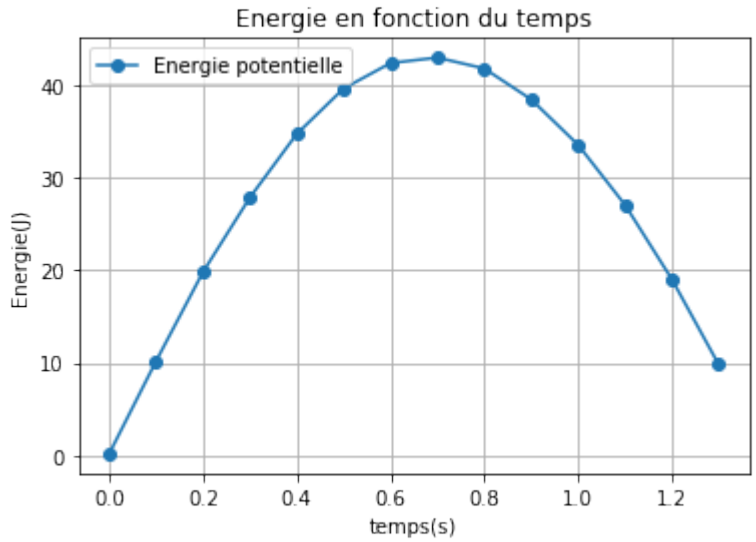
```
    plt.plot(abscisse,ordonnee,'o-',label=etiquette)
    plt.grid()
    plt.legend()
    plt.xlabel(abs)
    plt.ylabel(ordo)
    plt.title(titre)
    plt.show()


#programme prinipal

#entrer les positions du système suivant les deux axes du repère.
X=np.array([0.01,0.41,0.91,1.39,1.86,2.33,2.81,3.33,3.79,4.31,4.83,5.36,5.86,6.34,
↪6.77])
#liste contenant les abscisses.
Y=np.array([0.01,0.52,1.01,1.42,1.77,2.02,2.16,2.19,2.13,1.96,1.71,1.38,0.97,0.50,
↪0])
#liste contenant les ordonnées.
# repérage temporel des positions à partir de la fréquence d'aquisition et du␣
↪nombre total de points.
temps=[]
for i in range(len(X)):
    t=0.1*i
    temps.append(t)
#initialisation des deux listes qui recevront les valeurs des. vitesses␣
↪instantanées suivant les deux axes.
vity=[]
vitx=[]
#initialisation de la liste recevant la valeur de la vitesse.
vinstantanee=[]
#initialisation des listes recevant les valeurs des differentes energies.
ecinetique=[]
epotentielle=[]
emecanique=[]
#calcul des énergies pour les différentes positions utilisation d'une boucle for.
for i in range(len(X)-1):
    derivation(Y[i],Y[i+1],temps[i],temps[i+1])
    vity.append(derivee)
    derivation (X[i], X[i+1],temps[i],temps[i+1])
    vitx.append(derivee)
    module(vitx[i],vity[i])
    vinstantanee.append(norme)
    energie_cinetique(norme,2)
    ecinetique.append(energiecinetique)
    energie_potentielle(Y[i],2)
    epotentielle.append(energiepotentielle)
    energie_mecanique()
    emecanique.append(energiemecanique)
# On ajuste le nombre d'éléments des deux listes en enlevant le dernier élément de␣
↪la liste des instants.
tps=temps[:-1]
#modifier les arguments des différents paramètres de ces fonctions pour obtenir l
↪'affichage désiré.Selon l'exemple ci-dessous.
representation_graphique(tps,emecanique,"energie mécanique",)
representation_graphique(tps,ecinetique,"Energie cinétique")
representation_graphique(tps,epotentielle,"Energie potentielle")
```


Energie en fonction du temps
energie mécanique
Energie(J)
temps(s)


Energie en fonction du temps
Energie cinétique
Energie(J)
temps(s)


Energie en fonction du temps
Energie potentielle
Energie(J)
temps(s)

### 2.4.4 Mouvement d'un satellite géostationnaire (version élève)

`Télécharger le pdf`

`Télécharger le notebook`

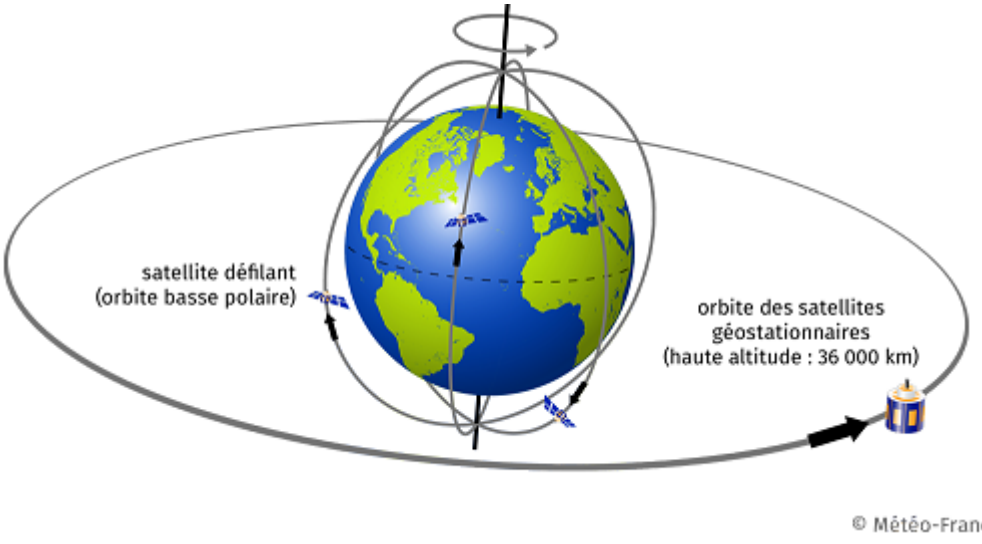
`Lancer le notebook sur binder (lent)`

**Document :**

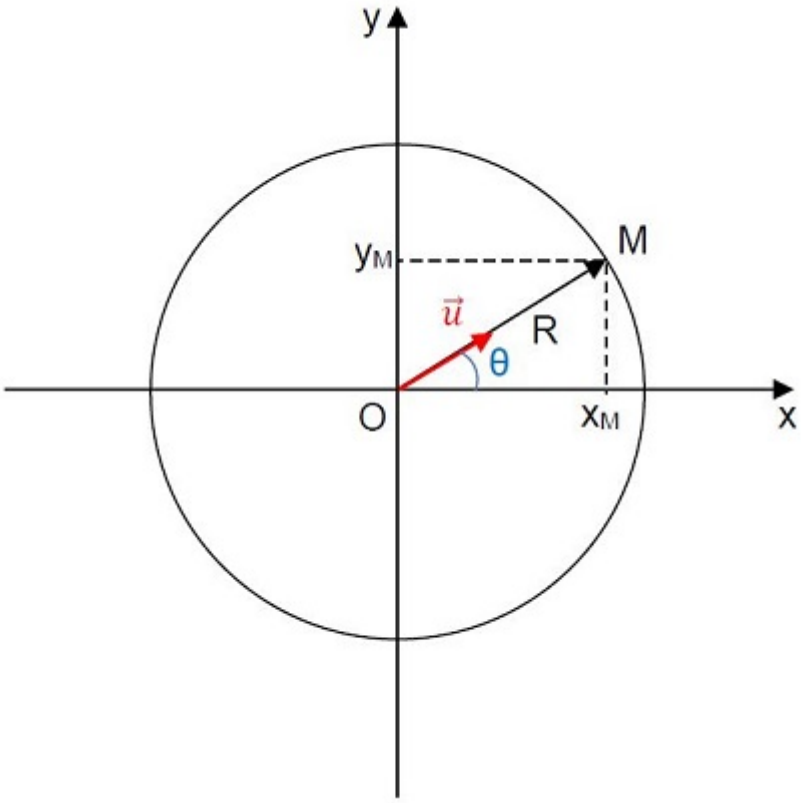
GOES-17 est le deuxième satellite de la génération actuelle de satellites météorologiques exploités par l'Administration nationale des océans et de l'atmosphère (NOAA). Il s'agit d'un satellite géostationnaire qui vise à fournir des images haute résolution visibles et infrarouges et des observations de la foudre sur plus de la moitié du globe. Le satellite a été lancé dans l'espace le 1er mars 2018 par un véhicule Atlas V (541) depuis la base aérienne de Cape Canaveral, en Floride. Il avait une masse de lancement de 5 192 kg (sa masse sèche (sans le carburant (ergols)) est de 2 857 kg). Le 12 mars, GOES-17 a rejoint GOES-16 (lancé en 2016) sur une orbite géosynchrone à 35 786 km au-dessus de la Terre (soit un rayon orbital de 42 164 km). GOES-17 est devenu opérationnel le 12 février 2019 sous le nom de GOES-West. Sa durée de vie utile prévue est de 15 ans.

L'orbite géosynchrone est une orbite géocentrique sur laquelle un satellite dit géostationnaire se déplace dans le même sens que la Terre (d'ouest en est) et dont la période orbitale est égale à la période de rotation sidérale de la Terre (soit environ 23 h 56 min 4 s). Un satellite géostationnaire reste donc toujours à la verticale d'un même lieu sur Terre.

source : Wikipédia (texte remanié)



**Rappels mathématiques** :




Les coordonnées cartésiennes du point M décrivant un cercle de rayon R centré sur l'origine O du repère sont :

$(x_M = R \times \cos\theta \; ; \; y_M = R \times \sin\theta)$

Le vecteur unitaire $\overrightarrow{u} = \frac{\overrightarrow{OM}}{OM} = \frac{\overrightarrow{OM}}{R}$ a pour coordonnées : $\overrightarrow{u} \begin{pmatrix} \frac{x_M}{R} = \cos\theta \\ \frac{y_M}{R} = \sin\theta \end{pmatrix}$

**Problématique : Comment la force d'attraction gravitationnelle exercée par la Terre sur le satellite GOES-17 influence la variation de son vecteur vitesse ?**

L'étude du mouvement du satellite GOES-17 aura lieu dans le référentiel géocentrique supposé galiléen auquel on associe un repère cartésien orthonormé fixe dont l'origine est au centre de la Terre.

1. Quelle est la nature du mouvement du satellite GOES-17 ? (Répondre dans la cellule ci-dessous en double-cliquant dessus si besoin).
2. En vous basant sur vos connaissances issues de la classe de seconde (en physique et en programmation), réfléchir aux différentes parties que comportera le programme permettant de répondre à la problématique. (Répondre dans la cellule ci-dessous en double-cliquant dessus si besoin).

[ ]:
```
# cellule 1 : import des bibliothèques

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

3. A l'aide du document, déterminer les valeurs du rayon R de la trajectoire et la période de révolution T du satellite (lignes 3 et 4 de la cellule 2).
4. Quelle valeur en radian prend l'angle $\theta$ lorsqu'il est parcouru par le segment OM en une période T ? (Répondre dans la cellule ci-dessous en double-cliquant dessus si besoin).

**note codage LaTeX** : double-cliquer sur cette cellule pour voir comment coder l'écriture des lettres grecques :

— théta : $\theta$
— pi : $\pi$

5. En déduire l'expression de l'angle $\theta$ en fonction du temps t et de la période T (ligne 8 de la cellule 2).
6. En déduire les coordonnées x et y de la position du satellite en vous aidant des rappels mathématiques (lignes 10 et 11 de la cellule 2).

**Note codage : la constante pi ainsi que les fonctions cos et sin sont fournies par la bibliothèque numpy** : - np.pi - np.cos() - np.sin()

```
[ ]: # cellule 2 : coordonnées de la position du satellite

R=    # rayon en m
T=       # période de révolution en s

t=np.arange(0,84164,500)

theta=

x=
y=
```

Afin de calculer les coordonnées du vecteur vitesse, **notées vx et vy** on crée une fonction **coordvit(u)** : - **u** est une liste et représente une des coordonnées (x ou y) du vecteur position. - **vu** est une liste et représente une des coordonnées (vx ou vy) du vecteur vitesse. - **vui** est la valeur à l'instant $t_i$ de la coordonnée étudiée du vecteur vitesse. La liste **vu** contiendra ces valeurs **vui**.

7. Compléter la ligne 6 de la cellule 3 permettant de calculer les valeurs **vui** prises par la coordonnée **vu** du vecteur vitesse à chaque instant du mouvement.

**Note codage : la variable t est déclarée dans le programme principal et est donc globale. Elle est ainsi reconnue au sein de toute fonction...**

**On rappelle que la :math :'i^{ème}' valeur d'une liste u est codée u[i]**

8. Ecrire les deux lignes de code permettant de calculer les valeurs des coordonnées **vx** et **vy** (lignes 10 et 11 de la cellule 3).

```
[ ]: # cellule 3 : coordonnées du vecteur vitesse du satellite

def coordvit(t,u):
    vu=[]
    for i in range (len(u)-1):
        vui=
        vu.append(vui)
    return(vu)

vx=
vy=
```

On appelle vecteur variation de vitesse au temps $t_i$, le vecteur : $\overrightarrow{\Delta v}(t_i) = \overrightarrow{v}(t_{i+1}) - \overrightarrow{v}(t_i)$.

On désire calculer les coordonnées notées dvx et dvy du vecteur $\overrightarrow{\Delta v}$

9. En vous basant sur le modèle de la fonction précédente, créer une fonction **coordvarvit(vu)** permettant de calculer les valeurs d'une coordonnée notée **dvu** du vecteur variation de vitesse (lignes 3 à 8 de la cellule 4).
10. Ecrire les deux lignes de code permettant de calculer les valeurs des coordonnées **dvx** et **dvy** (lignes 10 et 11 de la cellule 4).

```
[ ]: # cellule 4 : coordonnées du vecteur variation de vitesse




dvx=
dvy=
```

11. A l'aide du document, déterminer la valeur de la masse m du satellite après avoir consommé 1000 kg de carburant (ligne 4 de la cellule 5).
12. Donner l'expression vectorielle de la force d'attraction gravitationnelle exercée par la Terre sur le satellite $\overrightarrow{F_{T/S}}$ en fonction de G, $M_T$, m, R et $\overrightarrow{u}$. (Répondre dans la cellule ci-dessous en double-cliquant dessus si besoin).

**note codage LaTeX** : double-cliquer sur cette cellule pour voir comment coder l'écriture :
- d'un vecteur : $\overrightarrow{vecteur}$ ou $\vec{vecteur}$
- d'une fraction : $\frac{numrateur}{dnominateur}$
- d'un indice : $x_{indice}$
- d'un exposant : $y^{exposant}$
- d'un signe x : $\times$

13. Déterminer les expressions des coordonnées Fx et Fy de la force d'attraction gravitationnelle exercée par la Terre sur le satellite en vous aidant de vos connaissances et des rappels mathématiques (lignes 7 et 8 de la cellule 5).

```
[ ]: # cellule 5 : coordonnées du vecteur
     # force d'attraction gravitationnelle

     MT=5.972*10**24           # masse de la Terre en kg
     m=                        # masse du satellite en kg
     G=6.67408*10**(-11)       # constante de gravitation universelle
                               # en m^3.kg^-1.s^-2

     Fx=
     Fy=
```

14. Compléter la ligne de code 4 permttant d'afficher la trajectoire du satellite.
15. Compléter les lignes de code 5, 6 et 9 (et éventuellement 10) afin d'afficher les légendes des axes et le titre du graphique.
16. Sur le modèle des lignes de code 13 et 14, compléter les lignes de code 15 et 16 permettant de tracer le vecteur force d'attraction gravitationnelle.

```
[ ]: # cellule 6 : Tracé du graphique permettant de visualiser
     # la trajectoire du satellite ainsi que  les
     # vecteurs variation de vitesse et force d'attraction gravitationnelle.

     plt.figure (figsize=(10,10))

     plt.xlabel()
     plt.ylabel()
     plt.xlim(-50000000,50000000)
     plt.ylim(-50000000,50000000)
     plt.title ()
```

(continues on next page)

(suite de la page précédente)

```
for k in range(len(dvx)):
    plt.arrow(x[k],y[k],50000*dvx[k],50000*dvy[k],
              facecolor='b',edgecolor='b',width=200000,
              head_width=1000000,length_includes_head=True)


plt.show()
```

17. Conclusion : Répondre à la problématique dans la cellule suivante en double-cliquant dessus si besoin.

Remarque : si la direction du vecteur force et celle du vecteur variation de vitesse sont trop différentes, c'est que les vecteurs vitesses moyennes sont trop différents des vecteurs vitesses instantanées. Il suffit de diminuer la durée entre deux positions successives du satellite afin de réduire cette différence. (cf. la définition mathématique de la dérivée)

### 2.4.5 Mouvement d'un satellite géostationnaire (version professeur)

`Télécharger le pdf`

`Télécharger le notebook`

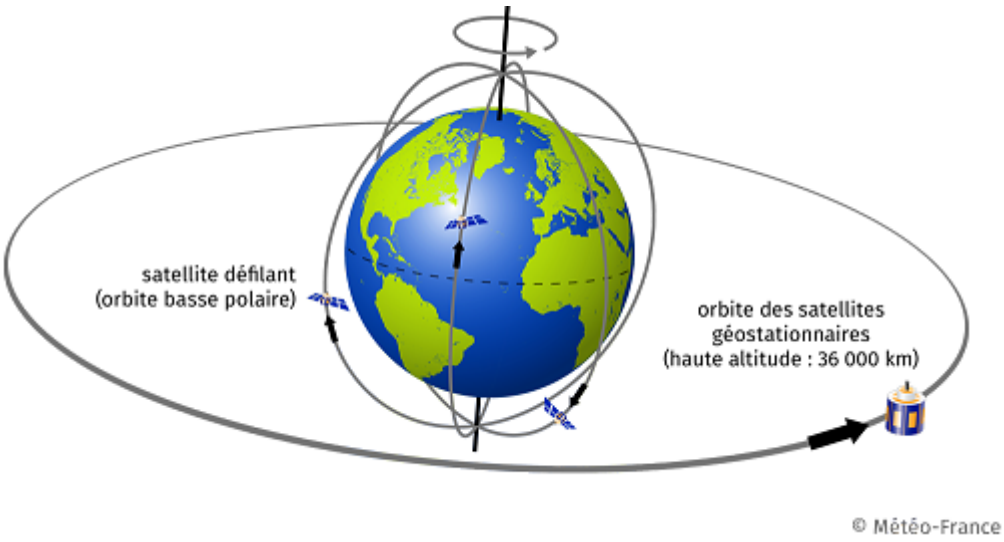
`Lancer le notebook sur binder (lent)`

**Document :**

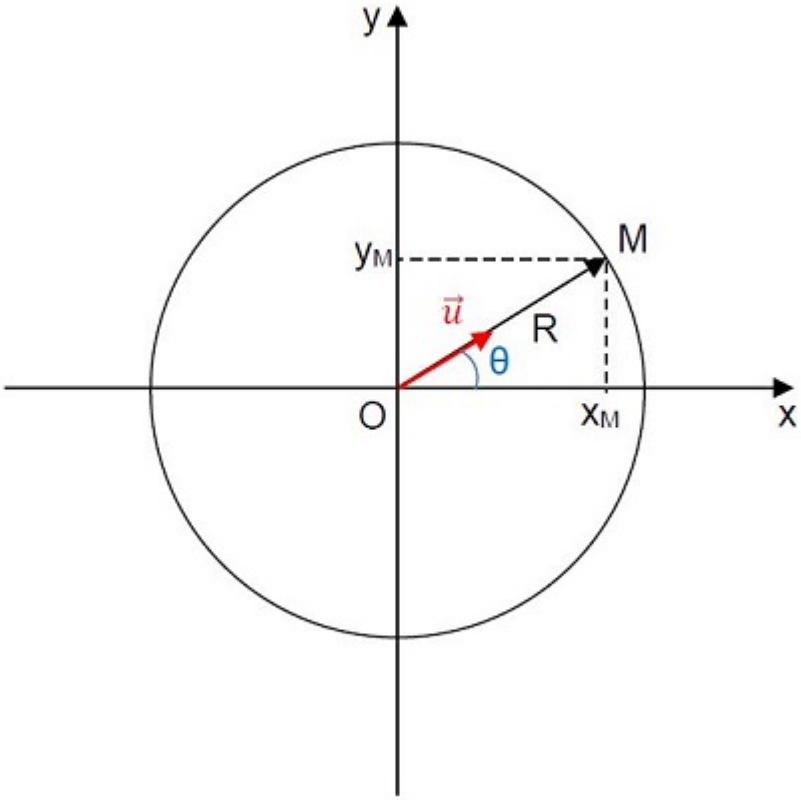
GOES-17 est le deuxième satellite de la génération actuelle de satellites météorologiques exploités par l'Administration nationale des océans et de l'atmosphère (NOAA). Il s'agit d'un satellite géostationnaire qui vise à fournir des images haute résolution visibles et infrarouges et des observations de la foudre sur plus de la moitié du globe. Le satellite a été lancé dans l'espace le 1er mars 2018 par un véhicule Atlas V (541) depuis la base aérienne de Cape Canaveral, en Floride. Il avait une masse de lancement de 5 192 kg (sa masse sèche (sans le carburant (ergols)) est de 2 857 kg). Le 12 mars, GOES-17 a rejoint GOES-16 (lancé en 2016) sur une orbite géosynchrone à 35 786 km au-dessus de la Terre (soit un rayon orbital de 42 164 km). GOES-17 est devenu opérationnel le 12 février 2019 sous le nom de GOES-West. Sa durée de vie utile prévue est de 15 ans.

L'orbite géosynchrone est une orbite géocentrique sur laquelle un satellite dit géostationnaire se déplace dans le même sens que la Terre (d'ouest en est) et dont la période orbitale est égale à la période de rotation sidérale de la Terre (soit environ 23 h 56 min 4 s). Un satellite géostationnaire reste donc toujours à la verticale d'un même lieu sur Terre.

source : Wikipédia (texte remanié)




**Rappels mathématiques** :




Les coordonnées cartésiennes du point M décrivant un cercle de rayon R centré sur l'origine O du repère sont :

$(x_M = R \times \cos\theta \; ; \; y_M = R \times \sin\theta)$

Le vecteur unitaire $\vec{u} = \frac{\overrightarrow{OM}}{OM} = \frac{\overrightarrow{OM}}{R}$ a pour coordonnées : $\vec{u} \begin{pmatrix} \frac{x_M}{R} = \cos\theta \\ \frac{y_M}{R} = \sin\theta \end{pmatrix}$

**Problématique : Comment la force d'attraction gravitationnelle exercée par la Terre sur le satellite GOES-17 influence la variation de son vecteur vitesse ?**

L'étude du mouvement du satellite GOES-17 aura lieu dans le référentiel géocentrique supposé galiléen auquel on associe un repère cartésien orthonormé fixe dont l'origine est au centre de la Terre.

1. Quelle est la nature du mouvement du satellite GOES-17 ? (Répondre dans la cellule ci-dessous en double-cliquant dessus si besoin).
2. En vous basant sur vos connaissances issues de la classe de seconde (en physique et en programmation), réfléchir aux différentes parties que comportera le programme permettant de répondre à la problématique. (Répondre dans la cellule ci-dessous en double-cliquant dessus si besoin).

— codage de la trajectoire du satellite (cellule 2)
— codage des coordonnées du vecteur vitesse (cellule 3)
— codage des coordonnées du vecteur variation de vitesse (cellule 4)
— codage des coordonnées du vecteur force d'attraction gravitationnelle (cellule 5)
— affichage d'un graphique représentant la trajectoire, le vecteur variation de vitesse du satellite et le vecteur force d'attraction gravitationnelle (cellule 6)

[1]:
```
# cellule 1 : import des bibliothèques

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

3. A l'aide du document, déterminer les valeurs du rayon R de la trajectoire et la période de révolution T du satellite (lignes 3 et 4 de la cellule 2).

4. Quelle valeur en radian prend l'angle $\theta$ lorsqu'il est parcouru par le segment OM en une période T ? (Répondre dans la cellule ci-dessous en double-cliquant dessus si besoin).

**note codage LaTeX** : double-cliquer sur cette cellule pour voir comment coder l'écriture des lettres grecques :
- théta : $\theta$
- pi : $\pi$

5. En déduire l'expression de l'angle $\theta$ en fonction du temps t et de la période T (ligne 8 de la cellule 2).
6. En déduire les coordonnées x et y de la position du satellite en vous aidant des rappels mathématiques (lignes 10 et 11 de la cellule 2).

**Note codage : la constante pi ainsi que les fonctions cos et sin sont fournies par la bibliothèque numpy** : - np.pi - np.cos() - np.sin()

[2]:
```
# cellule 2 : coordonnées de la position du satellite

R=42164000   # rayon en m
T=84164      # période de révolution en s

t=np.arange(0,84164,500)

theta=2*np.pi/T*t

x=R*np.cos(2*np.pi/T*t)
y=R*np.sin(2*np.pi/T*t)
```

Afin de calculer les coordonnées du vecteur vitesse, **notées vx et vy** on crée une fonction **coordvit(u)** :
- **u** est une liste et représente une des coordonnées (x ou y) du vecteur position.
- **vu** est une liste et représente une des coordonnées (vx ou vy) du vecteur vitesse.
- **vui** est la valeur à l'instant $t_i$ de la coordonnée étudiée du vecteur vitesse. La liste **vu** contiendra ces valeurs **vui**.

7. Compléter la ligne 6 de la cellule 3 permettant de calculer les valeurs **vui** prises par la coordonnée **vu** du vecteur vitesse à chaque instant du mouvement.

**Note codage : la variable t est déclarée dans le programme principal et est donc globale. Elle est ainsi reconnue au sein de toute fonction...**

**On rappelle que la :math :'i^{ème}' valeur d'une liste u est codée u[i]**

8. Ecrire les deux lignes de code permettant de calculer les valeurs des coordonnées **vx** et **vy** (lignes 10 et 11 de la cellule 3).

[3]:
```
# cellule 3 : coordonnées du vecteur vitesse du satellite

def coordvit(t,u):
    vu=[]
    for i in range (len(u)-1):
        vui=(u[i+1]-u[i])/(t[i+1]-t[i])
        vu.append(vui)
    return(vu)

vx=coordvit(t,x)
vy=coordvit(t,y)
```

On appelle vecteur variation de vitesse au temps $t_i$, le vecteur : $\overrightarrow{\Delta v}(t_i) = \overrightarrow{v}(t_{i+1}) - \overrightarrow{v}(t_i)$.

On désire calculer les coordonnées notées dvx et dvy du vecteur $\overrightarrow{\Delta v}$

9. En vous basant sur le modèle de la fonction précédente, créer une fonction **coordvarvit(vu)** permettant de calculer les valeurs d'une coordonnée notée **dvu** du vecteur variation de vitesse (lignes 3 à 8 de la cellule 4).

10. Ecrire les deux lignes de code permettant de calculer les valeurs des coordonnées **dvx** et **dvy** (lignes 10 et 11 de la cellule 4).

[4]:
```
# cellule 4 : coordonnées du vecteur variation de vitesse

def coordvarvit(vu):
    dvu=[]
    for j in range (len(vu)-1):
        dvuj=vu[j+1]-vu[j]
        dvu.append(dvuj)
    return(dvu)

dvx=coordvarvit(vx)
dvy=coordvarvit(vy)
```

11. A l'aide du document, déterminer la valeur de la masse m du satellite après avoir consommé 1000 kg de carburant (ligne 4 de la cellule 5).
12. Donner l'expression vectorielle de la force d'attraction gravitationnelle exercée par la Terre sur le satellite $\overrightarrow{F_{T/S}}$ en fonction de G, $M_T$, m, R et $\vec{u}$. (Répondre dans la cellule ci-dessous en double-cliquant dessus si besoin).

$\overrightarrow{F_{T/S}} = -G \times \frac{M_T \times m}{R^2} \times \vec{u}$

**note codage LaTeX** : double-cliquer sur cette cellule pour voir comment coder l'écriture :
- d'un vecteur : $\overrightarrow{vecteur}$ ou $\vec{vecteur}$
- d'une fraction : $\frac{numrateur}{dnominateur}$
- d'un indice : $x_{indice}$
- d'un exposant : $y^{exposant}$
- d'un signe x : $\times$

13. Déterminer les expressions des coordonnées Fx et Fy de la force d'attraction gravitationnelle exercée par la Terre sur le satellite en vous aidant de vos connaissances et des rappels mathématiques (lignes 7 et 8 de la cellule 5).

[5]:
```
# cellule 5 : coordonnées du vecteur
# force d'attraction gravitationnelle

MT=5.972*10**24          # masse de la Terre en kg
m=4192                   # masse du satellite en kg
G=6.67408*10**(-11)      # constante de gravitation universelle
                         # en m^3.kg^-1.s^-2

Fx=(G*MT*m/(R**2))*(-x/R)
Fy=(G*MT*m/(R**2))*(-y/R)
```

14. Compléter la ligne de code 4 permttant d'afficher la trajectoire du satellite.
15. Compléter les lignes de code 5, 6 et 9 (et éventuellement 10) afin d'afficher les légendes des axes et le titre du graphique.
16. Sur le modèle des lignes de code 13 et 14, compléter les lignes de code 15 et 16 permettant de tracer le vecteur force d'attraction gravitationnelle.

[6]:
```
# cellule 6 : Tracé du graphique permettant de visualiser
# la trajectoire du satellite ainsi que  les
# vecteurs variation de vitesse et force d'attraction gravitationnelle.

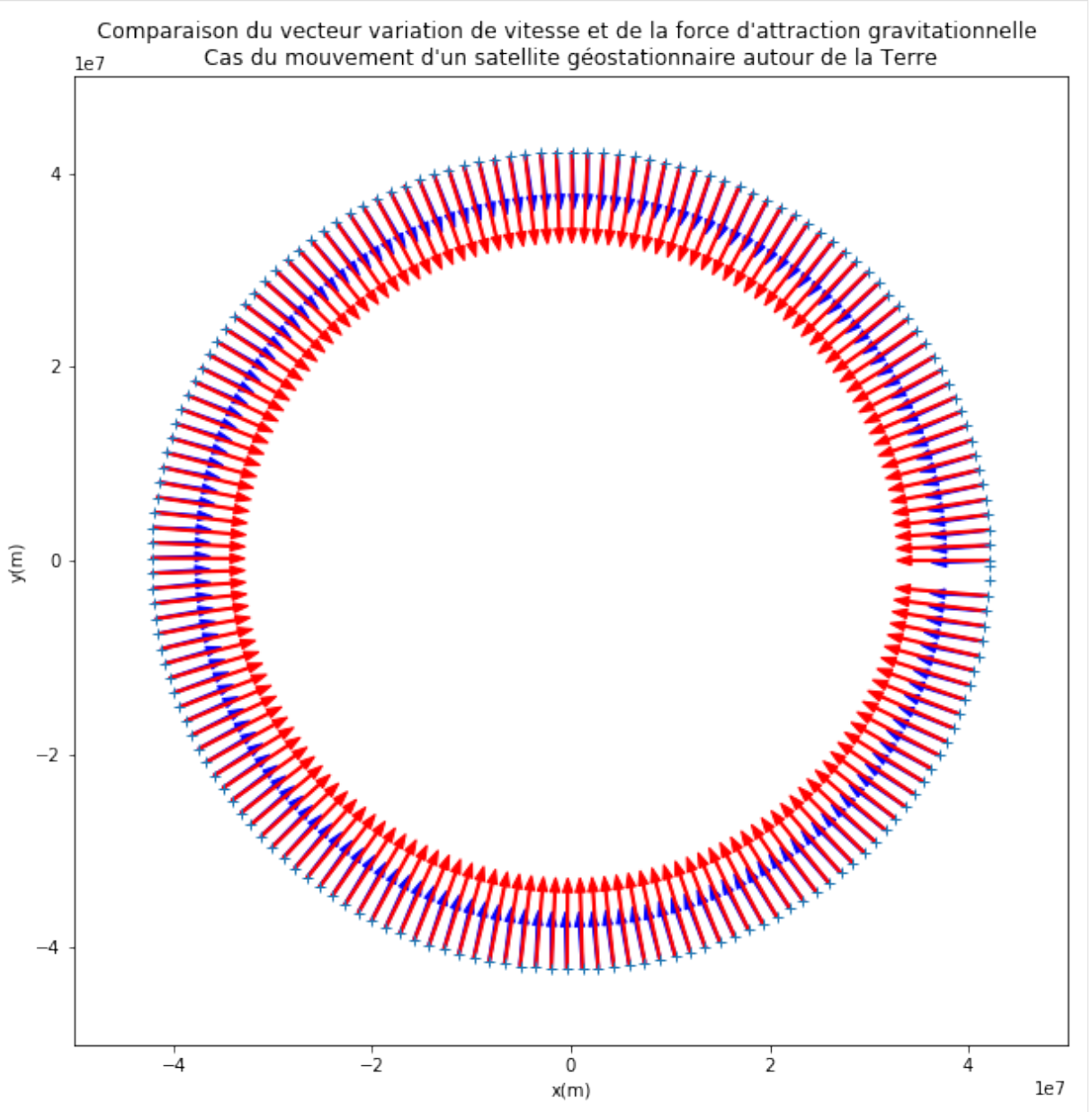
plt.figure (figsize=(10,10))
plt.plot(x,y,'+')
plt.xlabel('x(m)')
```

(continues on next page)

(suite de la page précédente)

```
plt.ylabel('y(m)')
plt.xlim(-50000000,50000000)
plt.ylim(-50000000,50000000)
plt.title ("Comparaison du vecteur variation de vitesse "
           "et de la force d'attraction gravitationnelle \n"
           "Cas du mouvement d'un satellite géostationnaire "
           "autour de la Terre")

for k in range(len(dvx)):
    plt.arrow(x[k],y[k],50000*dvx[k],50000*dvy[k],
              facecolor='b',edgecolor='b',width=200000,
              head_width=1000000,length_includes_head=True)
    plt.arrow(x[k],y[k],10000*Fx[k],10000*Fy[k],
              facecolor='r',edgecolor='r',width=200000,
              head_width=1000000,length_includes_head=True)
plt.show()
```




17. Conclusion : Répondre à la problématique dans la cellule suivante en double-cliquant dessus si besoin.

Remarque : si la direction du vecteur force et celle du vecteur variation de vitesse sont trop différentes, c'est que les vecteurs vitesses moyennes sont trop différents des vecteurs vitesses instantanées. Il suffit de diminuer la durée entre deux positions successives du satellite afin de réduire cette différence. (cf. la définition mathématique de la dérivée)

[ ]:

### 2.4.6 Etude de l'évolution d'un système chimique (version élève)

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Ce programme permet d'étudier l'évolution des quantités de matière des réactifs et produits d'une réaction dont l'équation est du type : aA + bB –> cC + dD où a, b, c et d sont les nombres stoechiométriques respectifs des espèces chimiques A, B, C et D.

Le programme doit tout d'abord demander les valeurs des nombres stoechiométriques, pour ensuite demander les quantités de matière initiales des réactifs A et B et des produits C et D.

[ ]:
```
# rattachement des librairies gérant les tracés
# de courbes et certains outils mathématiques
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

**Entrée des nombres stoechiométriques**

Sur le modèle de la ligne de code 2, ajouter les lignes de code nécessaires (lignes 3, 4 et 5) pour entrer les valeurs des nombres stoechiométriques b, c et d.

[ ]:
```
#Nombres stoechiométriques
a=2   #par exemple !! Donc à adapter...
```

[ ]:
```
# Affichage de l'équation de la réaction
print("l'équation étudiée est du type : ",
      a," A    +    ",b," B    -->     ",
      c," C      +    ",d," D")
```

**Entrée des valeurs de quantités de matière initiales**

Les valeurs des quantités de matière initiales des réactifs et des produits (exprimées en mole) seront stockées dans des variables notées n_0

(ex : nA_0 pour l'espèce chimique A).

Sur le modèle de la ligne de code 2, ajouter les lignes de code nécessaires (lignes 3, 4 et 5) pour entrer les quantités de matière initiales des autres espèces chimiques en jeu. Attention de bien entrer les valeurs en mol ! Vous pourrez par exemple taper 2.5e-3 pour 2,5 mmol

[ ]:
```
# Quantités de matières initiales
nA_0 =
nB_0 =
```

```
[ ]: #Initialisation des variables

     # Initialisation de la chaine de caractère correspondant
     # au réactif limitant
     Rlimitant =''

     # Avancement initial
     x=0

     # Pas d'avancement (on augmentera progressivement x de la valeur dx)
     dx=0.001

     # Création des listes contenant les quantités de matière
     # et initialisation de ces listes avec la valeur initiale
     nA=[nA_0]
     nB=[nB_0]
     nC=[nC_0]
     nD=[nD_0]

     # Création et nitialisation de la liste contenant l'avancement
     X=[x]
```

**Calculs des quantités de matière en cours d'avancement**

Sur le modèle de la ligne 5, écrire les lignes de code 6, 7 et 8 permettant de calculer les quantités de matière du réactif B, ainsi que des produits C et D.

NOTE CODAGE : l'instruction « nA.append(nA_0-a*x) » permet d'ajouter la valeur indiquée entre parenthèses à la fin de la liste nA.

**Détermination du réactif limitant**

Compléter les tests des lignes de code 11 et 12 en choisissant parmi : <0, <=0, >0 et >=0.

Compléter la ligne de code 13 en choisissant l'opérateur logique adéquat parmi : and (ET logique) et or (OU logique).

NOTE CODAGE : l'indice -1 permet d'avoir accès à la dernière valeur de la liste.

**Affichage du nom du réactif limitant et de l'avancement maximal**

Il sera intéressant de modifier en ligne 18, le nombre de chiffres après la virgule afin de respecter le nombre de chiffres significatifs pour l'avancement x.

```
[ ]: # Calculs des quantités de matière en cours d'avancement
     while nA[-1] > 0 and nB[-1] > 0:
         x=x+dx
         X.append(x)
         nA.append(nA_0-a*x)




     #Détermination du réactif limitant
     if nA[-1]                    : Rlimitant = 'A'
     if nB[-1]                    : Rlimitant = 'B'
     if nA[-1]<=0          nB[-1]<=0 :
         Rlimitant='A et B : le mélange est stoechiometrique'

     #Affichage des résultats
     print('Le réactif limitant est ',Rlimitant,
           '\n Avancement maximum : ','{0:.4f}'.format(x),
           'mol' )
     #{0:.4f} permet d'afficher un nombre arrondi à
     # 4 chiffres après la virgule (ici).
```

#### Affichage des courbes permettant de suivre l'évolution des quantités de matière

La ligne de code 2 ci-dessous permet d'afficher le graphe de l'évolution de la quantité de matière de A en fonction de l'avancement X.

Compléter les lignes 3, 4 et 5 pour afficher les courbes correspondant aux évolutions des quantités de matière des espèces chimiques B, C et D en fonction de l'avancement.

NOTE CODAGE : la commande plt.plot peut être enrichie de divers arguments (comme ici avec r- = r pour red et - pour ligne) :

- Couleur : r (red), k (black), b (blue), y (yellow), g (green)
- Marqueur : o (gros point), - (ligne), . (pointillé), x (croix), s (square), v (triangle)
- lw signifie linewidth (largeur de la ligne)

```
[ ]: plt.figure(figsize=(10,10))
     plt.plot(X,nA,'r-',lw=1,label='nA')



     plt.grid()
     plt.xlabel('x (mol)')
     plt.ylabel('n (mol)')
     plt.legend()
     plt.show()
```

#### Modélisation des droites obtenues

Les lignes de code suivantes vont permettre de modéliser chacune des 4 droites obtenues sur le graphe ci-dessus. Ces droites sont de type linéaire ou affine et peuvent être modélisées avec un polynôme de degré 1 : mx+p (où x est à la puissance 1). Les résultats des quatre modélisations sont ensuite affichés pour analyse.

Compléter les lignes de code 2, 3 et 4 (sur le modèle de la ligne 1) pour modéliser les courbes concernant l'évolution des quantités de matière des espèces chimiques B, C et D.

NOTE CODAGE : .0f en lignes 8, 13, 18 et 23 signifie qu'il y aura 1 seul chiffre significatif (pas de chiffre après la virgule), alors que .3f en lignes 9, 14, 19 et 24 signifie qu'il y aura 3 chiffres après la virgule.

```
[ ]: Amodel=np.polyfit(X, nA,1)




     print ("la droite représentant l'évolution de nA"
            " en fonction de x a pour équation : nA = ",
            '{0:.0f}'.format(Amodel[0]),'{:5}'.format("x +"),
            "{0:.3f}".format(Amodel[1]) )

     print ("la droite représentant l'évolution de nB"
            " en fonction de x a pour équation : nB = ",
            "{0:.0f}".format(Bmodel[0]),'{:5}'.format("x +"),
            "{0:.3f}".format(Bmodel[1]) )

     print ("la droite représentant l'évolution de nC"
            " en fonction de x a pour équation : nC = ",
            "{0:.0f}".format(Cmodel[0]),'{:5}'.format("x +"),
            "{0:.3f}".format(Cmodel[1]) )

     print ("la droite représentant l'évolution de nD"
            " en fonction de x a pour équation : nD = ",
            "{0:.0f}".format(Dmodel[0]),'{:5}'.format("x +"),
            "{0:.3f}".format(Dmodel[1]))
```

Commenter les équations des courbes modélisées....

### 2.4.7 Etude de l'évolution d'un système chimique (version professeur)

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Ce programme permet d'étudier l'évolution des quantités de matière des réactifs et produits d'une réaction dont l'équation est du type : aA + bB –> cC + dD où a, b, c et d sont les nombres stoechiométriques respectifs des espèces chimiques A, B, C et D.

Le programme doit tout d'abord demander les valeurs des nombres stoechiométriques, pour ensuite demander les quantités de matière initiales des réactifs A et B et des produits C et D.

[1]:
```
# rattachement des librairies gérant les tracés
# de courbes et certains outils mathématiques
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

**Entrée des nombres stoechiométriques**

Sur le modèle de la ligne de code 2, ajouter les lignes de code nécessaires (lignes 3, 4 et 5) pour entrer les valeurs des nombres stoechiométriques b, c et d.

[2]:
```
#Nombres stoechiométriques
a=2   #par exemple !! Donc à adapter...
b=1
c=1
d=3
```

[3]:
```
# Affichage de l'équation de la réaction
print("l'équation étudiée est du type : ",
      a," A    +    ",b," B    -->     ",
      c," C      +    ",d," D")
```

```
l'équation étudiée est du type :  2  A    +     1  B    ->      1  C      +     3
↪D
```

**Entrée des valeurs de quantités de matière initiales**

Les valeurs des quantités de matière initiales des réactifs et des produits (exprimées en mole) seront stockées dans des variables notées n_0

(ex : nA_0 pour l'espèce chimique A).

Sur le modèle de la ligne de code 2, ajouter les lignes de code nécessaires (lignes 3, 4 et 5) pour entrer les quantités de matière initiales des autres espèces chimiques en jeu. Attention de bien entrer les valeurs en mol ! Vous pourrez par exemple taper 2.5e-3 pour 2,5 mmol

[4]:
```
# Quantités de matières initiales
nA_0 = 2.5e-3    #par exemple !!
nB_0 = 5e-3
nC_0 = 0
nD_0 = 0
```

[5]:
```
#Initialisation des variables

# Initialisation de la chaine de caractère correspondant
# au réactif limitant
Rlimitant =''

# Avancement initial
```

(continues on next page)

(suite de la page précédente)

```
x=0

# Pas d'avancement (on augmentera progressivement x de la valeur dx)
dx=0.001

# Création des listes contenant les quantités de matière
# et initialisation de ces listes avec la valeur initiale
nA=[nA_0]
nB=[nB_0]
nC=[nC_0]
nD=[nD_0]

# Création et nitialisation de la liste contenant l'avancement
X=[x]
```

**Calculs des quantités de matière en cours d'avancement**

Sur le modèle de la ligne 5, écrire les lignes de code 6, 7 et 8 permettant de calculer les quantités de matière du réactif B, ainsi que des produits C et D.

NOTE CODAGE : l'instruction « nA.append(nA_0-a*x) » permet d'ajouter la valeur indiquée entre parenthèses à la fin de la liste nA.

**Détermination du réactif limitant**

Compléter les tests des lignes de code 11 et 12 en choisissant parmi : <0, <=0, >0 et >=0.

Compléter la ligne de code 13 en choisissant l'opérateur logique adéquat parmi : and (ET logique) et or (OU logique).

NOTE CODAGE : l'indice -1 permet d'avoir accès à la dernière valeur de la liste.

**Affichage du nom du réactif limitant et de l'avancement maximal**

Il sera intéressant de modifier en ligne 18, le nombre de chiffres après la virgule afin de respecter le nombre de chiffres significatifs pour l'avancement x.

[6]:
```
# Calculs des quantités de matière en cours d'avancement
while nA[-1] > 0 and nB[-1] > 0 :
    x=x+dx
    X.append(x)
    nA.append(nA_0-a*x)
    nB.append(nB_0-b*x)
    nC.append(nC_0+c*x)
    nD.append(nD_0+d*x)

#Détermination du réactif limitant
if nA[-1]  <=0                  : Rlimitant = 'A'
if nB[-1]  <=0                  : Rlimitant = 'B'
if nA[-1]  <=0   and   nB[-1]<=0 :
    Rlimitant='A et B : le mélange est stoechiometrique'

#Affichage des résultats
print('Le réactif limitant est ',Rlimitant,
      '\n Avancement maximum : ','{0:.4f}'.format(x),
      'mol' )
#{0:.4f} permet d'afficher un nombre arrondi à
# 4 chiffres après la virgule (ici).
```

```
Le réactif limitant est  A
 Avancement maximum :  0.0020 mol
```

**Affichage des courbes permettant de suivre l'évolution des quantités de matière**

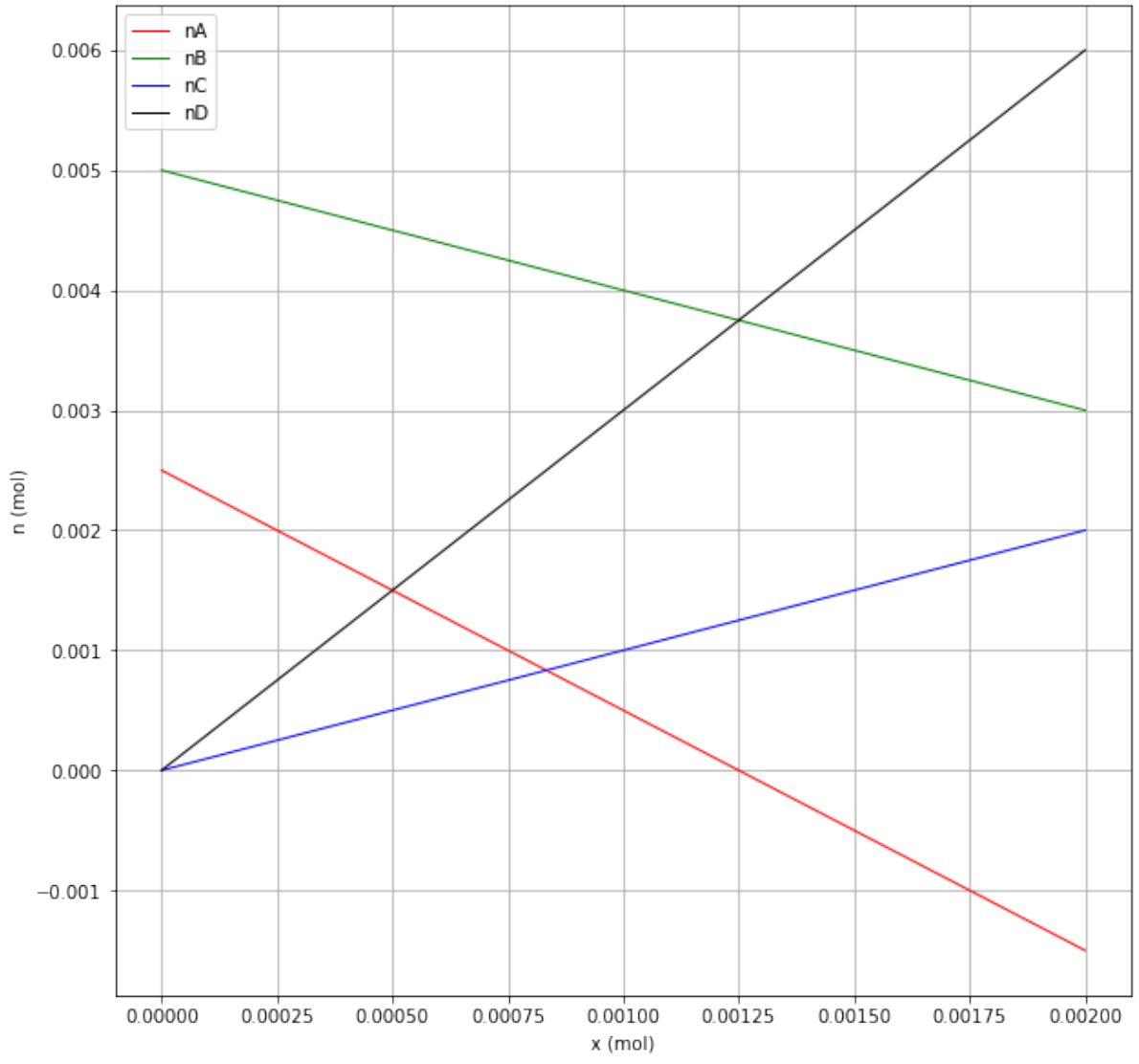
La ligne de code 2 ci-dessous permet d'afficher le graphe de l'évolution de la quantité de matière de A en fonction de l'avancement X.

Compléter les lignes 3, 4 et 5 pour afficher les courbes correspondant aux évolutions des quantités de matière des espèces chimiques B, C et D en fonction de l'avancement.

NOTE CODAGE : la commande plt.plot peut être enrichie de divers arguments (comme ici avec r- = r pour red et - pour ligne) :

- Couleur : r (red), k (black), b (blue), y (yellow), g (green)
- Marqueur : o (gros point), - (ligne), . (pointillé), x (croix), s (square), v (triangle)
- lw signifie linewidth (largeur de la ligne)

[7]:
```
plt.figure(figsize=(10,10))
plt.plot(X,nA,'r-',lw=1,label='nA')
plt.plot(X,nB,'g-',lw=1,label='nB')
plt.plot(X,nC,'b-',lw=1,label='nC')
plt.plot(X,nD,'k-',lw=1,label='nD')
plt.grid()
plt.xlabel('x (mol)')
plt.ylabel('n (mol)')
plt.legend()
plt.show()
```




**Modélisation des droites obtenues**

Les lignes de code suivantes vont permettre de modéliser chacune des 4 droites obtenues sur le graphe ci-dessus. Ces droites sont de type linéaire ou affine et peuvent être modélisées avec un polynôme de degré 1 : mx+p (où x est à la puissance 1). Les résultats des quatre modélisations sont ensuite affichés pour analyse.

Compléter les lignes de code 2, 3 et 4 (sur le modèle de la ligne 1) pour modéliser les courbes concernant l'évolution des quantités de matière des espèces chimiques B, C et D.

NOTE CODAGE : .0f en lignes 8, 13, 18 et 23 signifie qu'il y aura 1 seul chiffre significatif (pas de chiffre après la virgule), alors que .3f en lignes 9, 14, 19 et 24 signifie qu'il y aura 3 chiffres après la virgule.

[8]:
```
Amodel=np.polyfit(X, nA,1)
Bmodel=np.polyfit(X, nB,1)
Cmodel=np.polyfit(X, nC,1)
Dmodel=np.polyfit(X, nD,1)

print ("la droite représentant l'évolution de nA"
       " en fonction de x a pour équation : nA = ",
       '{0:.0f}'.format(Amodel[0]),'{:5}'.format("x +"),
       "{0:.3f}".format(Amodel[1]) )

print ("la droite représentant l'évolution de nB"
       " en fonction de x a pour équation : nB = ",
       "{0:.0f}".format(Bmodel[0]),'{:5}'.format("x +"),
       "{0:.3f}".format(Bmodel[1]) )

print ("la droite représentant l'évolution de nC"
       " en fonction de x a pour équation : nC = ",
       "{0:.0f}".format(Cmodel[0]),'{:5}'.format("x +"),
       "{0:.3f}".format(Cmodel[1]) )

print ("la droite représentant l'évolution de nD"
       " en fonction de x a pour équation : nD = ",
       "{0:.0f}".format(Dmodel[0]),'{:5}'.format("x +"),
       "{0:.3f}".format(Dmodel[1]))
```

```
la droite représentant l'évolution de nA en fonction de x a pour équation : nA = ␣
↪-2 x +   0.002
la droite représentant l'évolution de nB en fonction de x a pour équation : nB = ␣
↪-1 x +   0.005
la droite représentant l'évolution de nC en fonction de x a pour équation : nC = ␣
↪1 x +   0.000
la droite représentant l'évolution de nD en fonction de x a pour équation : nD = ␣
↪3 x +   0.000
```

Commenter les équations des courbes modélisées....

### 2.4.8 Etude de l'influence de l'amplitude et de la période pour un signal périodique (version élève)

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Nous allons étudier un signal sinusoïdal. Un tel signal se répète identique à lui-même tous les 2 $\pi$, au bout d'une durée T (période en s).

Son évolution au cours du temps t se traduit par la fonction mathématique : A.sin((2$\pi$/T).t)

où A est l'amplitude

Comme le temps t ne peut pas être continu, il faut le discrétiser, c'est-à-dire calculer t pour des valeurs entières, multiples d'une petite durée appelée *période d'échantillonnage* et notée Te.

Te doit être suffisamment petit par rapport à T.

Ce qui donne l'expression mathématique suivante : A.sin((2π/T).i.Te)

Avec i prenant des valeurs entières.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# cette fonction permet d'afficher un graphique
# à un emplacement précis de la fenêtre graphique.
# Ainsi, on peut afficher plusieurs sous-graphiques
# sur la même fenêtre.

def affichage_graphique(n,y,l,yl) :
# Déclaration du nombre d'emplacements dans la fenêtre
    plt.subplot(3,1,n)
# Affichage de la courbe
    plt.plot(t,y,'k-',lw=1,label=n)
 # Impose les bornes min et max sur l'axe des ordonnées
    plt.ylim(-2,2)
    plt.grid()
    plt.xlabel('t (s)')
    plt.ylabel(yl)
    plt.legend()

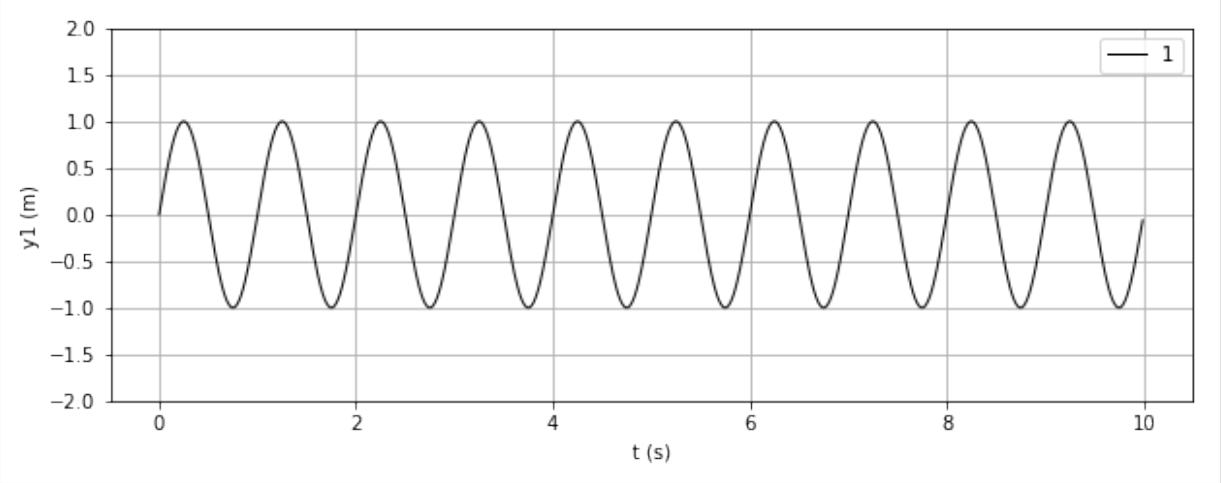
# Déclaration des variables
Ymax=1   # amplitude en m
T=1      # période en s
Te=0.01   # période d'échantillonnage en s
# Création des listes (vides) qui contiendront les valeurs
# du temps et des amplitudes
t=[]
y1=[]

# Boucle permettant de parcourir toutes les valeurs du temps
# discrétisé.
for i in range (0,1000) :
# La méthode append permet de rajouter une valeur en fin
# de la liste t
    t.append(i*Te)
# la fonction sinus est contenue dans la bibliothèque numpy
# la constante pi est contenue dans la bibliothèque numpy
    y1.append(Ymax*np.sin((2*np.pi/T)*i*Te))


# création du graphique

# création de la fenêtre graphique
plt.figure(2,figsize=(10,12))
# appel de la fonction gérant l'affichage du sous-graphique
affichage_graphique(1,y1,"courbe de référence","y1 (m)")

plt.show()
```




**Etude préalable :**

En tenant compte des renseignements donnés lignes 24, 25 et 33, répondez aux questions suivantes :

1. Combien d'échantillons temporels aura-t-on ?
2. Par quel calcul simple aurait-on pu prévoir la durée du signal créé ?
3. Par quel calcul simple aurait-on pu prévoir le nombre de périodes affichées ?
4. Combien d'échantillons temporels a-t-on par période ?

Nous souhaitons étudier l'influence des paramètres A et T sur l'évolution temporelle du signal sinusoïdal.

Pour cela, nous avons déjà écrit en lignes 25 et 37 du programme ci-dessous, la création d'un signal sinusoïdal de référence noté y1.

Nous avons également déjà déclaré les listes y2 et y3 sur les lignes de code 26 et 27.

Sur le modèle de la ligne 37, compléter la ligne 38 de manière à créer un signal sinusoïdal nommé y2 d'amplitude deux fois plus grande que y1.

Puis, toujours sur le même modèle, compléter la ligne 39 de manière à créer un signal sinusoïdal nommé y3 de période deux fois plus grande que y1.

Nous souhaitons de plus, afficher ces trois signaux en trois graphiques situés l'un au-dessous de l'autre. Nous allons pour cela utiliser la méthode plt.subplot(nombre de lignes, nombre de colonnes, index) de la bibliothèque matplotlib.pyplot as plt.

L'affichage est géré par une fonction nommée affichage_graphique qui a besoin d'un certain nombre de paramètres (fournis entre parenthèses) pour fonctionner correctement.

Sur le modèle de la ligne 43, écrire la ligne de code nécessaire à l'affichage de y2.

Puis, toujours sur le modèle de la ligne 43, écrire la ligne de code nécessaire à l'affichage de y3.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def affichage_graphique(n,y,l,yl) :
# Déclaration du nombre d'emplacements dans la fenêtre
    plt.subplot(3,1,n)
# Affichage de la courbe
    plt.plot(t,y,'k-',lw=1,label=n)
# Impose les bornes min et max sur l'axe des ordonnées
    plt.ylim(-2,2)
    plt.grid()
    plt.xlabel('t (s)')
    plt.ylabel(yl)
    plt.legend()
```

(continues on next page)

(suite de la page précédente)

```
# Déclaration des variables
Ymax=1    # amplitude en m
T=1       # période en s
Te=0.01    # période d'échantillonnage en s

# Création des listes (vides) qui contiendront les valeurs
# du temps et des amplitudes
t=[]
y1=[]
y2=[]
y3=[]

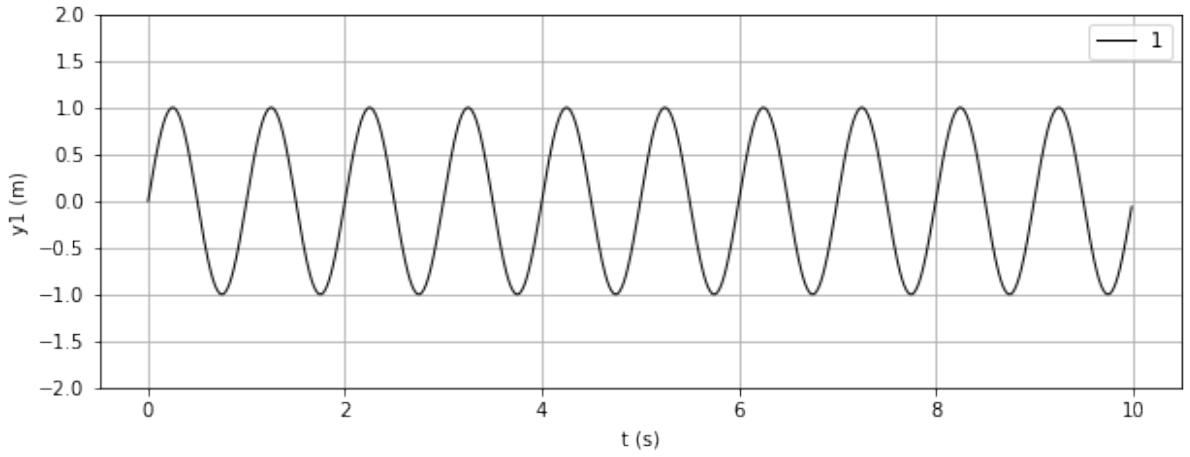
# Boucle permettant de parcourir toutes les valeurs du temps discrétisé.
for i in range (0,1000) :
# La méthode append permet de rajouter une valeur en fin de la liste t
    t.append(i*Te)
# la fonction sinus est contenue dans la bibliothèque numpy
# la constante pi est contenue dans la bibliothèque numpy
# On aurait pu aussi utiliser la bibliothèque math pour y avoir accés
# à l'aide des fonctions math.sin() et math.pi
    y1.append(Ymax*np.sin((2*np.pi/T)*i*Te))



    # création de la fenêtre graphique
plt.figure(2,figsize=(10,12))
affichage_graphique(1,y1,"courbe de référence","y1 (m)")



plt.show()
```




### 2.4.9 Etude de l'influence de l'amplitude et de la période pour un signal périodique (version professeur)

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Nous allons étudier un signal sinusoïdal. Un tel signal se répète identique à lui-même tous les 2 $\pi$, au bout d'une durée T (période en s).

Son évolution au cours du temps t se traduit par la fonction mathématique : A.sin((2π/T).t)

où A est l'amplitude

Comme le temps t ne peut pas être continu, il faut le discrétiser, c'est à dire calculer t pour des valeurs entières, multiples d'une petite durée appelée *période d'échantillonnage* et notée Te.

Te doit être suffisamment petit par rapport à T.

Ce qui donne l'expression mathématique suivante : A.sin((2π/T).i.Te)

Avec i prenant des valeurs entières.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# cette fonction permet d'afficher un graphique
# à un emplacement précis de la fenêtre graphique.
# Ainsi, on peut afficher plusieurs sous-graphiques
# sur la même fenêtre.

def affichage_graphique(n,y,l,yl) :
# Déclaration du nombre d'emplacements dans la fenêtre
    plt.subplot(3,1,n)
# Affichage de la courbe
    plt.plot(t,y,'k-',lw=1,label=n)
 # Impose les bornes min et max sur l'axe des ordonnées
    plt.ylim(-2,2)
    plt.grid()
    plt.xlabel('t (s)')
    plt.ylabel(yl)
    plt.legend()

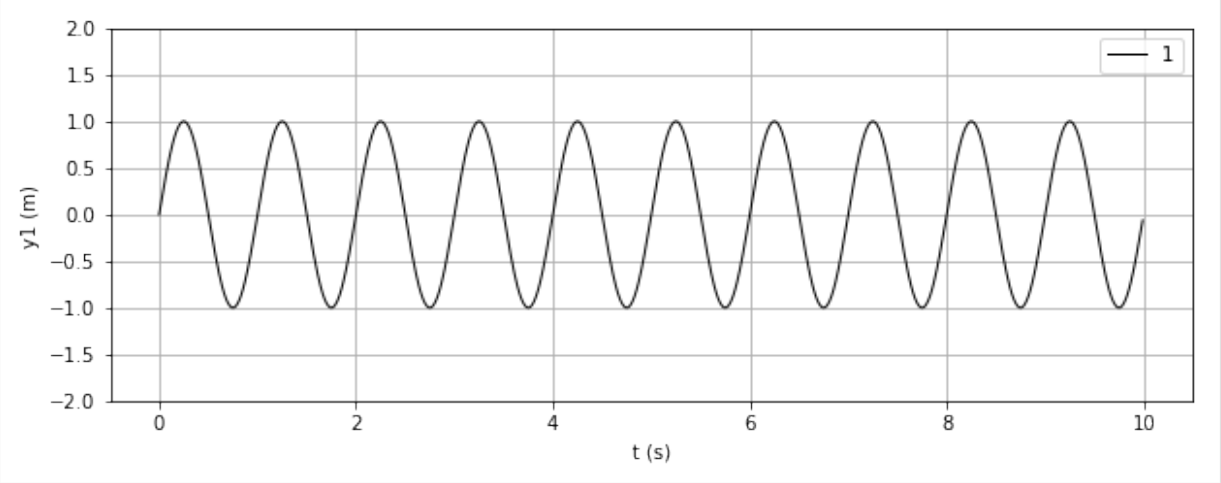
# Déclaration des variables
Ymax=1   # amplitude en m
T=1      # période en s
Te=0.01   # période d'échantillonnage en s
# Création des listes (vides) qui contiendront les valeurs
# du temps et des amplitudes
t=[]
y1=[]

# Boucle permettant de parcourir toutes les valeurs du temps
# discrétisé.
for i in range (0,1000) :
# La méthode append permet de rajouter une valeur en fin
# de la liste t
    t.append(i*Te)
# la fonction sinus est contenue dans la bibliothèque numpy
# la constante pi est contenue dans la bibliothèque numpy
    y1.append(Ymax*np.sin((2*np.pi/T)*i*Te))


# création du graphique

# création de la fenêtre graphique
plt.figure(2,figsize=(10,12))
# appel de la fonction gérant l'affichage du sous-graphique
affichage_graphique(1,y1,"courbe de référence","y1 (m)")

plt.show()
```




**Etude préalable :**

En tenant compte des renseignements donnés lignes 24, 25 et 33 répondez aux questions suivantes : 1. Combien d'échantillons temporels aura-t-on ? 2. Par quel calcul simple aurait-on pu prévoir la durée du signal créé ? 3. Par quel calcul simple aurait-on pu prévoir le nombre de périodes affichées ? 4. Combien d'échantillons temporels a-t-on par période ?

Réponses : 1. 1000 (voir boucle) 2. durée = nb d'échantillons * Te = 1000 * 0.01 = 10 s 3. nb périodes = durée / T = 10/1 = 10 périodes 4. T/Te = 1/0.01 = 100 échantillons temporels par période

Nous souhaitons étudier l'influence des paramètres A et T sur l'évolution temporelle du signal sinusoïdal.

Pour cela, nous avons déjà écrit en lignes 25 et 37 du programme ci-dessous, la création d'un signal sinusoïdal de référence noté y1.

Nous avons également déjà déclaré les listes y2 et y3 sur les lignes de code 26 et 27.

Sur le modèle de la ligne 37, compléter la ligne 38 de manière à créer un signal sinusoïdal nommé y2 d'amplitude deux fois plus grande que y1.

Puis, toujours sur le même modèle, compléter la ligne 39 de manière à créer un signal sinusoïdal nommé y3 de période deux fois plus grande que y1.

Nous souhaitons de plus, afficher ces trois signaux en trois graphiques situés l'un au-dessous de l'autre. Nous allons pour cela utiliser la méthode plt.subplot(nombre de lignes, nombre de colonnes, index) de la bibliothèque matplotlib.pyplot as plt.

L'affichage est géré par une fonction nommée affichage_graphique qui a besoin d'un certain nombre de paramètres (fournis entre parenthèses) pour fonctionner correctement.

Sur le modèle de la ligne 43, écrire la ligne de code nécessaire à l'affichage de y2.

Puis, toujours sur le modèle de la ligne 43, écrire la ligne de code nécessaire à l'affichage de y3.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def affichage_graphique(n,y,l,yl) :
# Déclaration du nombre d'emplacements dans la fenêtre
    plt.subplot(3,1,n)
# Affichage de la courbe
    plt.plot(t,y,'k-',lw=1,label=n)
# Impose les bornes min et max sur l'axe des ordonnées
    plt.ylim(-2,2)
    plt.grid()
    plt.xlabel('t (s)')
```

(continues on next page)

(suite de la page précédente)

```
    plt.ylabel(yl)
    plt.legend()

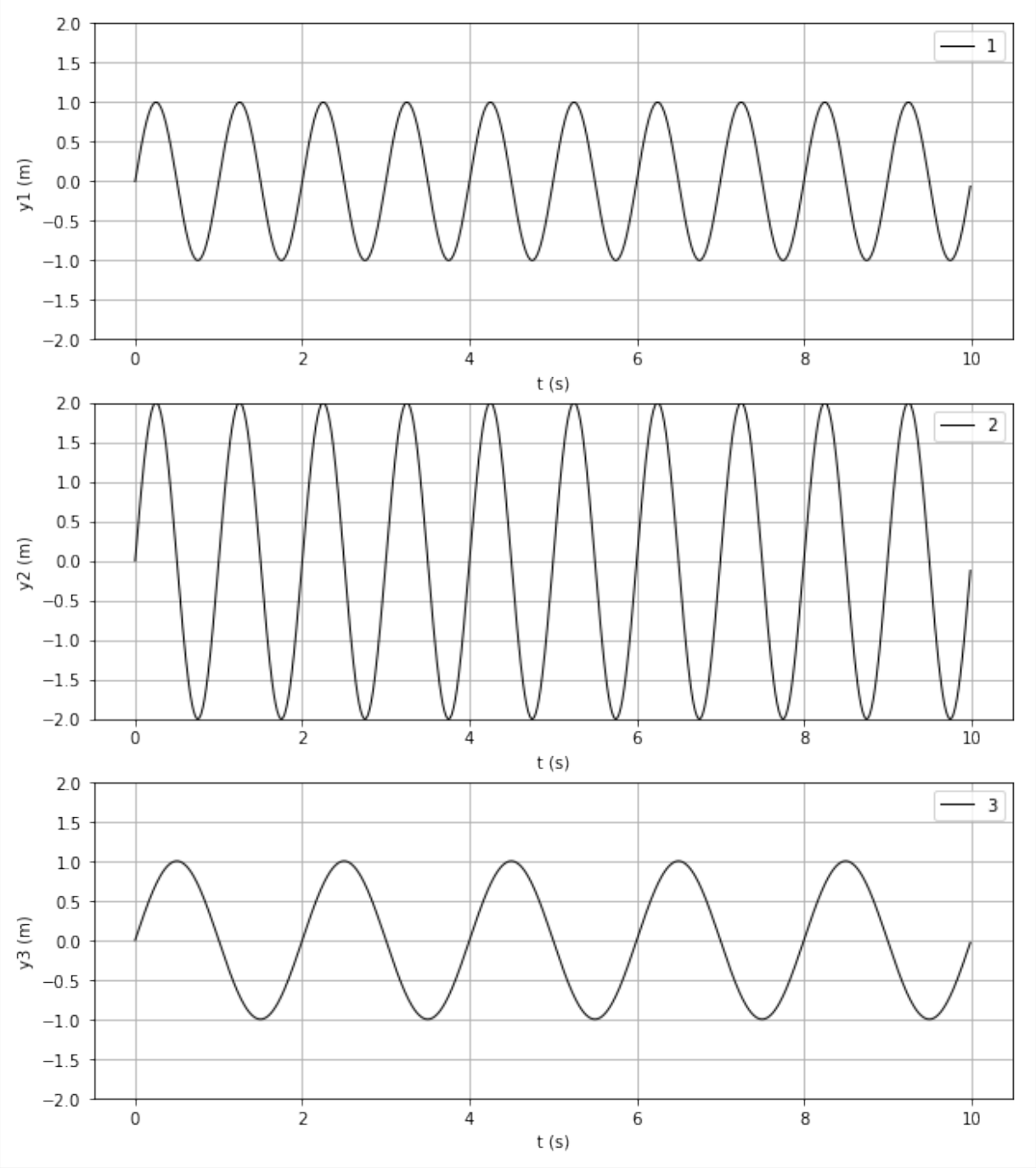
# Déclaration des variables
Ymax=1    # amplitude en m
T=1       # période en s
Te=0.01    # période d'échantillonnage en s

# Création des listes (vides) qui contiendront les valeurs
# du temps et des amplitudes
t=[]
y1=[]
y2=[]
y3=[]

# Boucle permettant de parcourir toutes les valeurs du temps discrétisé.
for i in range (0,1000) :
# La méthode append permet de rajouter une valeur en fin de la liste t
    t.append(i*Te)
# la fonction sinus est contenue dans la bibliothèque numpy
# la constante pi est contenue dans la bibliothèque numpy
# On aurait pu aussi utiliser la bibliothèque math pour y avoir accés
# à l'aide des fonctions math.sin() et math.pi
    y1.append(Ymax*np.sin((2*np.pi/T)*i*Te))
    y2.append(2*Ymax*np.sin((2*np.pi/T)*i*Te))
    y3.append(Ymax*np.sin((2*np.pi/(2*T))*i*Te))

    # création de la fenêtre graphique
plt.figure(2,figsize=(10,12))
affichage_graphique(1,y1,"courbe de référence","y1 (m)")
affichage_graphique(2,y2,"amplitude doublée","y2 (m)")
affichage_graphique(3,y3,"période doublée","y3 (m)")

plt.show()
```




## 2.5 Activités pour la seconde

### 2.5.1 Simulation de décroissance radioactive

Simulation de décroissance radioactive par le jet de dés

code sous licence creative commun CC BY-NC-SA BY Alexis Dendiével

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

[1]:
```
# programme de simulation de décroissance radioactive
# par le jet de dés
# lignes de code sous licence creative commun CC BY-NC-SA
# BY Alexis Dendiével

%matplotlib inline

# importations
import matplotlib.pyplot as plt
from random import randint

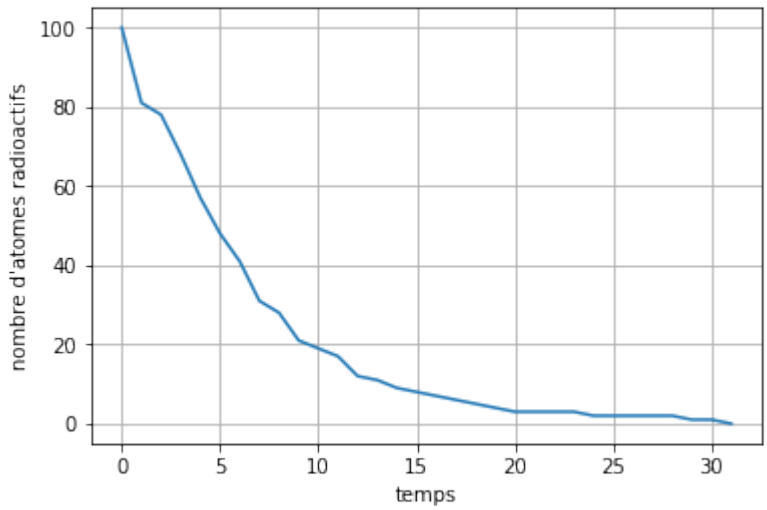
# entrée du nombre d'atomes radioactifs pour la simulation
n = int(input("pour combien d'atomes radioactifs voulez-vous faire la simulation?
↪: "))

# initialisation des données
nombrelance = 0
temps = [0]
radioactifs = [n]

# coeur du programme
while n > 0:
        desintegration = 0
        for i in range(n):
            tirage = randint(1,6)
            if tirage == 6:
                desintegration = desintegration + 1
        n = n - desintegration
        nombrelance = nombrelance + 1
        temps.append(nombrelance)
        radioactifs.append(n)

# affichage
plt.plot(temps, radioactifs)
plt.grid()
plt.xlabel("temps")
plt.ylabel("nombre d'atomes radioactifs")
plt.show()
plt.close()
```

```
pour combien d'atomes radioactifs voulez-vous faire la simulation? : 100
```

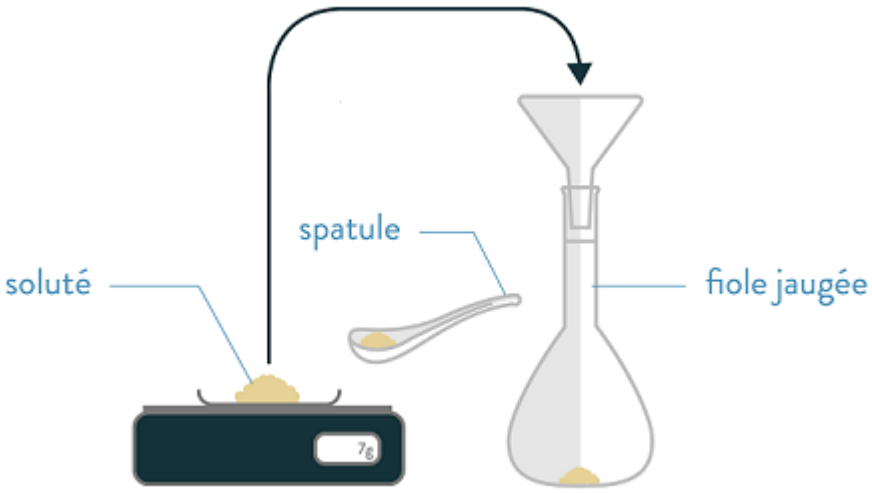



[ ]:

[ ]:

### 2.5.2 Préparation d'une solution par dissolution (version élève)

Contexte : Un technicien de laboratoire aurait besoin d'un petit programme en Python afin de calculer facilement la masse m de soluté à peser pour fabriquer une solution de concentration en soluté apporté C et de volume V. Aidez-le à réaliser ce petit programme ! !




source : https://www.schoolmouv.fr

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Pour commencer, il faut définir les différentes variables utiles pour faire le calcul. Compléter les deux cellules vides ci-dessous en vous aidant du modèle de la cellule de la masse molaire. Ne pas oublier d'exécuter chaque cellule pour vérifier que votre code est correct !

[ ]:
```
# ligne de code permettant de définir la variable M et
# de lui attribuer une valeur.

M=58.5   # masse molaire en g/mol


# ligne de code permettant d'afficher la valeur de la
# variable M

print ('M =',M,'g/mol')


# ligne de code permettant d'afficher la valeur de la
# variable M en écriture décimale avec une décimale

print('M ={0:.1f}'.format(M),'g/mol')


# ligne de code permettant d'afficher la valeur de la
# variable en écriture scientifique avec deux décimales
# donc trois chiffres significatifs
```

(continues on next page)

(suite de la page précédente)

```
print('M ={0:.2e}'.format(M),'g/mol')
```

[ ]:

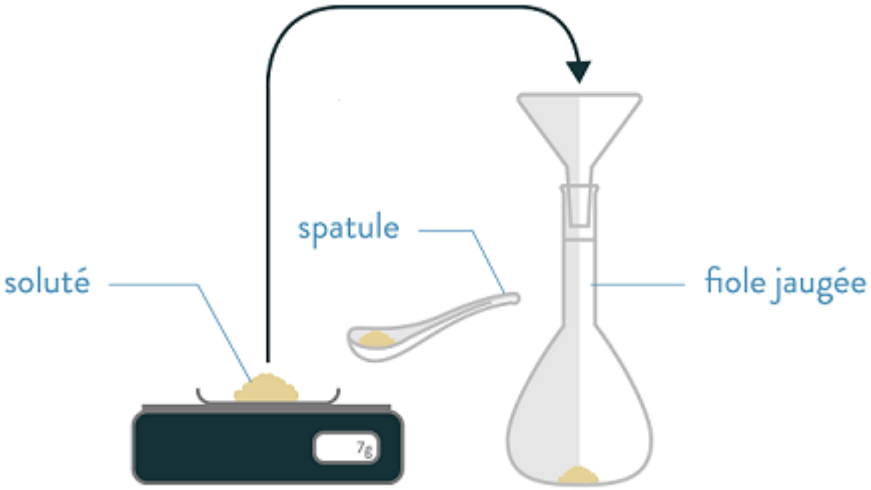
[ ]:

Maintenant, il reste à écrire dans la cellule suivante les lignes de code permettant de calculer puis d'afficher la valeur de la masse de soluté en g.

[ ]:

### 2.5.3 Préparation d'une solution par dissolution (version professeur)

Contexte : Un technicien de laboratoire aurait besoin d'un petit programme en Python afin de calculer facilement la masse m de soluté à peser pour fabriquer une solution de concentration en soluté apporté C et de volume V. Aidez-le à réaliser ce petit programme ! !




source : https://www.schoolmouv.fr

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Pour commencer, il faut définir les différents objets utiles pour faire le calcul. Compléter les deux cellules vides ci-dessous en vous aidant du modèle de la cellule de la masse molaire. Ne pas oublier d'exécuter chaque cellule pour vérifier que votre code est correct !

[1]:
```
# ligne de code permettant de définir la variable M et
# de lui attribuer une valeur.

M=58.5   # masse molaire en g/mol


# ligne de code permettant d'afficher la valeur de la
# variable M

print ('M =',M,'g/mol')
```

(continues on next page)

(suite de la page précédente)

```
# ligne de code permettant d'afficher la valeur de la
# variable M en écriture décimale avec une décimale

print('M ={0:.1f}'.format(M),'g/mol')


# ligne de code permettant d'afficher la valeur de la
# variable M en écriture scientifique avec deux décimales
# donc trois chiffres significatifs

print('M ={0:.2e}'.format(M),'g/mol')
```

```
M = 58.5 g/mol
M =58.5 g/mol
M =5.85e+01 g/mol
```

[2]:
```
V=0.25          # volume en L
print('V ={0:.4f}'.format(V),'L')
```

```
V =0.2500 L
```

[3]:
```
C=0.1          # Concentration molaire en mol/l
print('C ={0:.1e}'.format(C),'mol/L')
```

```
C =1.0e-01 mol/L
```

Maintenant, il reste à écrire dans la cellule suivante les lignes de code permettant de calculer puis d'afficher la valeur de la masse de soluté en g.

[4]:
```
m=C*M*V
print('m ={0:.1e}'.format(m),'g')
```

```
m =1.5e+00 g
```

### 2.5.4 La loi d'Ohm (version élève, linregress et sans fonction)

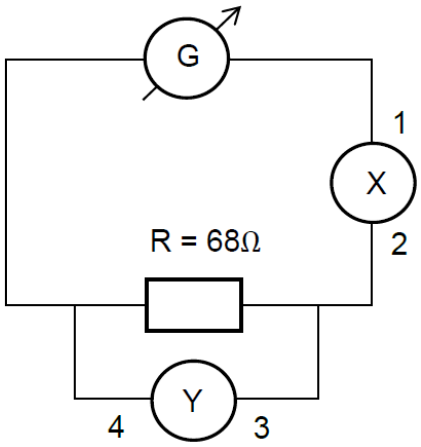
Mathilde, élève de 2nde, souhaite tracer la caractéristique d'un dipôle ohmique, c'est-à-dire la courbe donnant les valeurs de la tension aux bornes du dipôle ohmique en fonction des valeurs de l'intensité du courant qui le traverse.

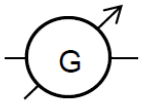
`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Elle a schématisé le circuit de son expérience :




Symbole d'un générateur dont on peut faire varier la tension.

1. Dans la cellule ci-dessous, indiquer la signification des symboles X et Y et le nom des bornes 1, 2, 3, 4.

X : 1 : 2 : Y : 3 : 4 :

Mathilde relève les mesures expérimentales suivantes :

| I (mA) | 0 | 25 | 50 | 75 | 100 | 125 |
|---|---|---|---|---|---|---|
| U (V) | 0 | 1,8 | 3,3 | 5,2 | 6,8 | 8,5 |

2. Aider Mathilde à coder la deuxième ligne du tableau de valeurs dans la cellule vide ci-dessous en vous aidant du code de la première ligne (attention les valeurs de l'intensité y ont été converties en ampère).

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[ ]:
```
# array signifie tableau en anglais
I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
print (I)
```

[ ]:
```

```

3. Mathilde veut maintenant afficher la caractéristique « intensité-tension » du dipôle ohmique en respectant les consignes suivantes :
   — axe des abscisses (horizontal) : Intensité I (mA)
   — axe des ordonnées (vertical) : Tension U(V)
   — points expérimentaux : croix + de couleur rouge
   — Titre : « Caractéristique Intensité-Tension d'un dipôle ohmique »

Les cellules ci-dessous contiennent chacune une ligne du code nécessaire à l'affichage de la caractéristique. Exécuter chaque cellule au fur et à mesure afin de comprendre leur utilité. Noter si besoin des commentaires dans les cellules laissées vides à cet effet.

[ ]:
```
fig = plt.figure(figsize=(12,10))
```

commentaire :

[ ]:
```
plt.plot(I,U)
```

commentaire :

[ ]:
```
plt.plot(I,U,'r+')
```

[ ]:
```
plt.plot(I,U,'r+')
plt.show()
```

commentaire :

[ ]:
```
plt.plot(I,U,'r+',label='U=f(I)')
plt.legend()
```

commentaire :

[ ]:
```
plt.xlabel("intensité I (A)")
```

commentaire :

[ ]:
```
plt.ylabel("tension U (V)")
```

commentaire :

[ ]:
```
plt.grid()
```

commentaire :

[ ]:
```
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
```

commentaire :

4. Exécutez maintenant le programme en entier !

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
U=np.array([0,1.7,3.4,5.1,6.8,8.5])
fig = plt.figure(figsize=(12,10))
plt.plot(I,U,'r+',label='U=f(I)')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
plt.show()
```

5. Il s'agit maintenant de modéliser la courbe obtenue.

5.1. Quelle est la forme de la courbe obtenue ?

5.2. Quelle est l'équation mathématique d'une telle courbe ?

5.3. Exécutez le programme ci-dessous permettant de modéliser la courbe obtenue par une droite.

[ ]:
```
from scipy import stats
slope, intercept, r_value, p_value, std_error = stats.linregress(I, U)
print ('slope {0:.2f}'.format(slope))
print('intercept {0:.2f}'.format(intercept))
Umodel = slope*I+intercept
print('U= {0:.2f}'.format(slope),'x I')
print ('Le coefficient de corrélation r vaut {0:.4f}'.format(r_value))
print('Les valeurs de la tension modélisée sont',Umodel)
```

5.3.1. Que représente l'objet slope ?

5.3.2. Que représente l'objet intercept ?

5.4. Affichez la droite modélisée grâce au programme ci-dessous.

```
[ ]: fig = plt.figure(figsize=(12,10))
plt.plot(I,U,'r+',label='U=f(I)')
plt.plot(I,Umodel,'b',label='modèle linéaire')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
plt.show()
```

5.4.1. La tension U et l'intensité I sont-elles proportionnelles ? Pourquoi ?

5.4.2. Que remarquez-vous à propos de la valeur du coefficient directeur de la droite ?

5.4.3. En déduire une formule appelée loi d'Ohm entre la tension U, l'intensité I et la résistance électrique R du dipôle ohmique.

### 2.5.5 La loi d'Ohm (version élève, polyfit et sans fonction)

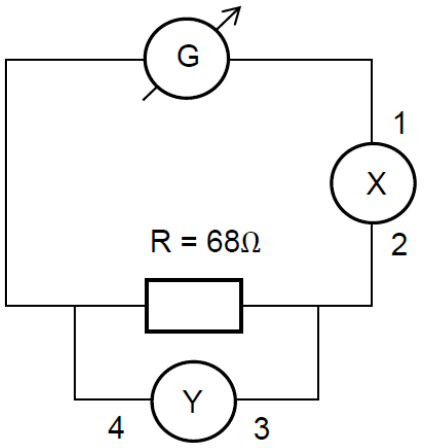
Mathilde, élève de 2nde, souhaite tracer la caractéristique d'un dipôle ohmique, c'est-à-dire la courbe donnant les valeurs de la tension aux bornes du dipôle ohmique en fonction des valeurs de l'intensité du courant qui le traverse.

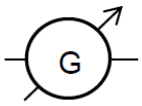
`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Elle a schématisé le circuit de son expérience :




Symbole d'un générateur dont on peut faire varier la tension.

1. Dans la cellule ci-dessous, indiquer la signification des symboles X et Y et le nom des bornes 1, 2, 3, 4.

X : 1 : 2 : Y : 3 : 4 :

Mathilde relève les mesures expérimentales suivantes :

| I (mA) | 0 | 25 | 50 | 75 | 100 | 125 |
|---|---|---|---|---|---|---|
| U (V) | 0 | 1,8 | 3,3 | 5,2 | 6,8 | 8,5 |

2. Aider Mathilde à coder la deuxième ligne du tableau de valeurs dans la cellule vide ci-dessous en vous aidant du code de la première ligne (attention les valeurs de l'intensité y ont été converties en ampère).

```
[ ]: import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline
```

```
[ ]: # array signifie tableau en anglais
     I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
     print (I)
```

```
[ ]:
```

3. Mathilde veut maintenant afficher la caractéristique « intensité-tension » du dipôle ohmique en respectant les consignes suivantes :
   — axe des abscisses (horizontal) : Intensité I (mA)
   — axe des ordonnées (vertical) : Tension U(V)
   — points expérimentaux : croix + de couleur rouge
   — Titre : « Caractéristique Intensité-Tension d'un dipôle ohmique »

Les cellules ci-dessous contiennent chacune une ligne du code nécessaire à l'affichage de la caractéristique. Exécuter chaque cellule au fur et à mesure afin de comprendre leur utilité. Noter si besoin des commentaires dans les cellules laissées vides à cet effet.

```
[ ]: fig = plt.figure(figsize=(12,10))
```

```
[ ]: plt.plot(I,U)
```

commentaire :

```
[ ]: plt.plot(I,U,'r+')
```

```
[ ]: plt.plot(I,U,'r+')
     plt.show()
```

commentaire :

```
[ ]: plt.plot(I,U,'r+',label='U=f(I)')
     plt.legend()
```

commentaire :

```
[ ]: plt.xlabel("intensité I (A)")
```

commentaire :

```
[ ]: plt.ylabel("tension U (V)")
```

commentaire :

```
[ ]: plt.grid()
```

commentaire :

```
[ ]: plt.title("Caractéristique Intensité-Tension "
               "d'un dipôle ohmique")
```

commentaire :

4. Exécutez maintenant le programme en entier !

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
U=np.array([0,1.7,3.4,5.1,6.8,8.5])
fig = plt.figure(figsize=(12,10))
plt.plot(I,U,'r+',label='U=f(I)')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension d'un "
          "dipôle ohmique")
plt.show()
```

5. Il s'agit maintenant de modéliser la courbe obtenue.

5.1. Quelle est la forme de la courbe obtenue ?

5.2. Quelle est l'équation mathématique d'une telle courbe ?

5.3. Exécutez le programme ci-dessous permettant de modéliser la courbe obtenue par une droite.

[ ]:
```
coeff=np.polyfit(I, U,1)
print ('{0:.1f}'.format(coeff[0]),
       '{0:.1f}'.format(coeff[1]))
Umodel = coeff[0]*I+coeff[1]
print('U={0:.1f}'.format(coeff[0]),'x I')
print('Les valeurs de la tension modélisée sont',Umodel)
```

5.3.1. Que représente coeff [0] ?

5.3.2. Que représente coeff[1] ?

5.4. Affichez la droite modélisée grâce au programme ci-dessous.

[ ]:
```
fig = plt.figure(figsize=(12,10))
plt.plot(I,U,'r+',label='U=f(I)')
plt.plot(I,Umodel,'b',label='modèle linéaire')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
plt.show()
```

5.4.1. La tension U et l'intensité I sont-elles proportionnelles ? Pourquoi ?

5.4.2. Que remarquez-vous à propos de la valeur du coefficient directeur de la droite ?

5.4.3. En déduire une formule appelée loi d'Ohm entre la tension U, l'intensité I et la résistance électrique R du dipôle ohmique.

### 2.5.6 La loi d'Ohm (version linregress et avec fonctions)

`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

| I (mA) | 0 | 25 | 50 | 75 | 100 | 125 |
|---|---|---|---|---|---|---|
| U (V) | 0 | 1,8 | 3,3 | 5,2 | 6,8 | 8,5 |

[1]:
```
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# création de la fonction modelisation
# modélisation par une droite d'équation
# y=ax+b (polynôme de degré 1)

def modelisation(x,y):
    slope, intercept, r_value, p_value, std_error = stats.linregress(x,y)
    ymodel = slope*x+intercept
    print ('U= {0:.1f}'.format(slope),'x I')
    print ('Le coefficient de corrélation r vaut {0:.4f}'.format(r_value))
    print('Les valeurs de la tension modélisée sont',ymodel)
    return (ymodel)
```

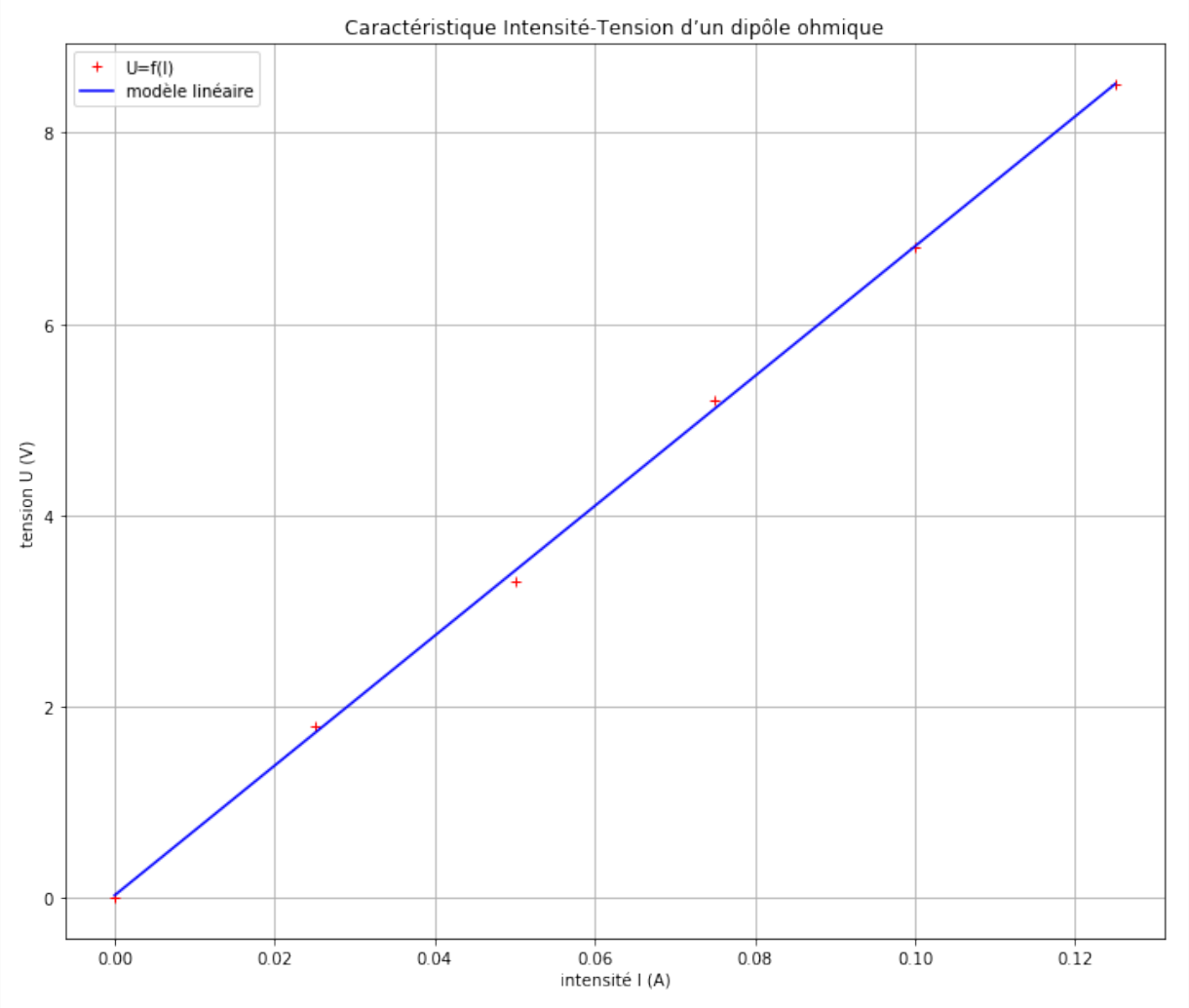
[3]:
```
# fonction permettant de tracer le graphique avec les points
# expérimentaux et la courbe obtenue après modélisation

def courbemodelisee (x,y,ymodel) :
    fig = plt.figure(figsize=(12,10))
    plt.plot(x,y,'r+',label='U=f(I)')
    plt.plot(x,ymodel,'b',label='modèle linéaire')
    plt.legend()
    plt.xlabel("intensité I (A)")
    plt.ylabel("tension U (V)")
    plt.grid()
    plt.title("Caractéristique Intensité-Tension d'un "
              "dipôle ohmique")
    plt.show()
```

[4]:
```
# tableaux numpy obligatoires à cause de l'opération vectorisée
# permettant de créer Umodel

I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
U=np.array([0,1.8,3.3,5.2,6.8,8.5])
Umodel=modelisation(I,U)
courbemodelisee(I,U,Umodel)
```

```
U= 67.9 x I
Le coefficient de corrélation r vaut 0.9997
Les valeurs de la tension modélisée sont [0.02380952 1.72095238 3.41809524 5.
↪1152381  6.81238095 8.50952381]
```




### 2.5.7 La loi d'Ohm (version professeur, linregress et sans fonction)

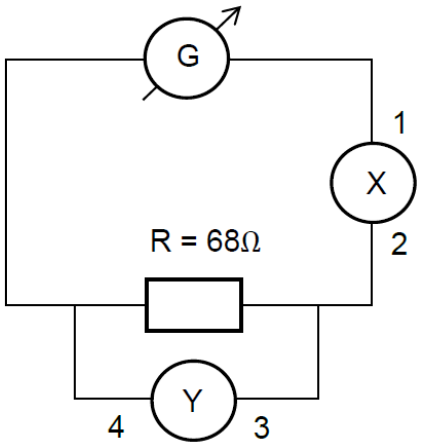
Mathilde, élève de 2nde, souhaite tracer la caractéristique d'un dipôle ohmique, c'est-à-dire la courbe donnant les valeurs de la tension aux bornes du dipôle ohmique en fonction des valeurs de l'intensité du courant qui le traverse.

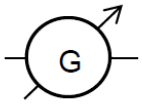
`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Elle a schématisé le circuit de son expérience :




Symbole d'un générateur dont on peut faire varier la tension.

1. Dans la cellule ci-dessous, indiquer la signification des symboles X et Y et le nom des bornes 1, 2, 3, 4.

X : 1 : 2 : Y : 3 : 4 :

Mathilde relève les mesures expérimentales suivantes :

| I (mA) | 0 | 25 | 50 | 75 | 100 | 125 |
|---|---|---|---|---|---|---|
| U (V) | 0 | 1,8 | 3,3 | 5,2 | 6,8 | 8,5 |

2. Aider Mathilde à coder la deuxième ligne du tableau de valeurs dans la cellule vide ci-dessous en vous aidant du code de la première ligne (attention les valeurs de l'intensité y ont été converties en ampère).

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# array signifie tableau en anglais
I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
print (I)
```

```
[0.    0.025 0.05  0.075 0.1   0.125]
```

[3]:
```
U=np.array([0,1.8,3.3,5.2,6.8,8.5])
print(U)
```

```
[0.  1.8 3.3 5.2 6.8 8.5]
```

3. Mathilde veut maintenant afficher la caractéristique « intensité-tension » du dipôle ohmique en respectant les consignes suivantes :
   — axe des abscisses (horizontal) : Intensité I (mA)
   — axe des ordonnées (vertical) : Tension U(V)
   — points expérimentaux : croix + de couleur rouge
   — Titre : « Caractéristique Intensité-Tension d'un dipôle ohmique »

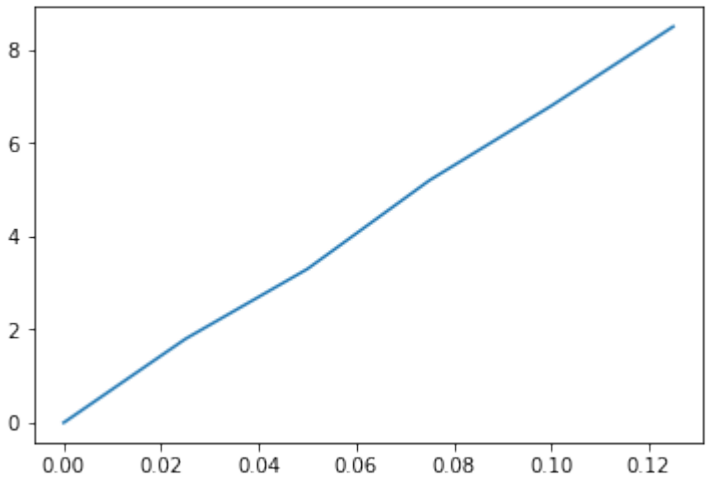
Les cellules ci-dessous contiennent chacune une ligne du code nécessaire à l'affichage de la caractéristique. Exécuter chaque cellule au fur et à mesure afin de comprendre leur utilité. Noter si besoin des commentaires dans les cellules laissées vides à cet effet.

[4]:
```
fig = plt.figure(figsize=(12,10))
```

```
<Figure size 864x720 with 0 Axes>
```

[5]:
```
plt.plot(I,U)
```

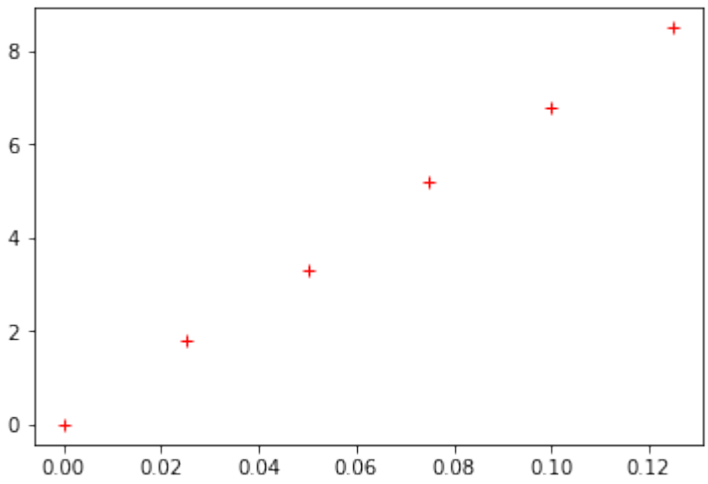
[5]:
```
[<matplotlib.lines.Line2D at 0x7f309d049b70>]
```



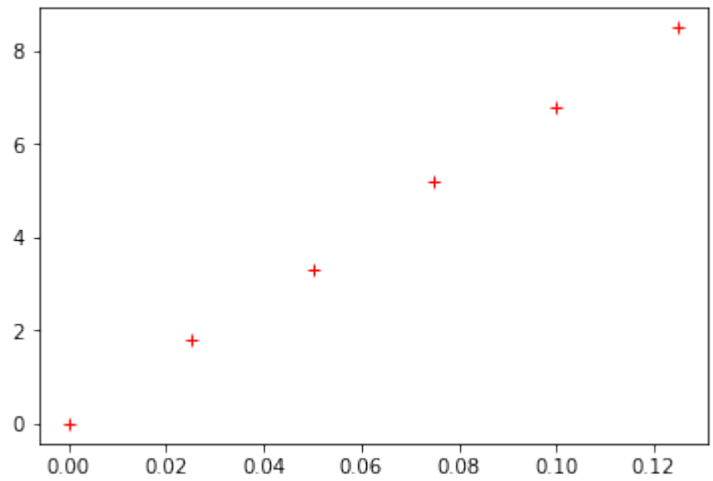

commentaire :

[6]:
```
plt.plot(I,U,'r+')
```

[6]:
```
[<matplotlib.lines.Line2D at 0x7f309cfd2fd0>]
```



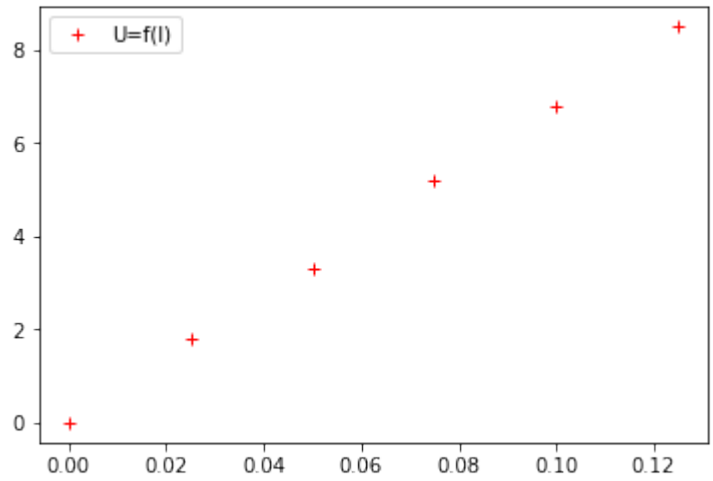

[7]:
```
plt.plot(I,U,'r+')
plt.show()
```




commentaire :

[8]:
```
plt.plot(I,U,'r+',label='U=f(I)')
plt.legend()
```

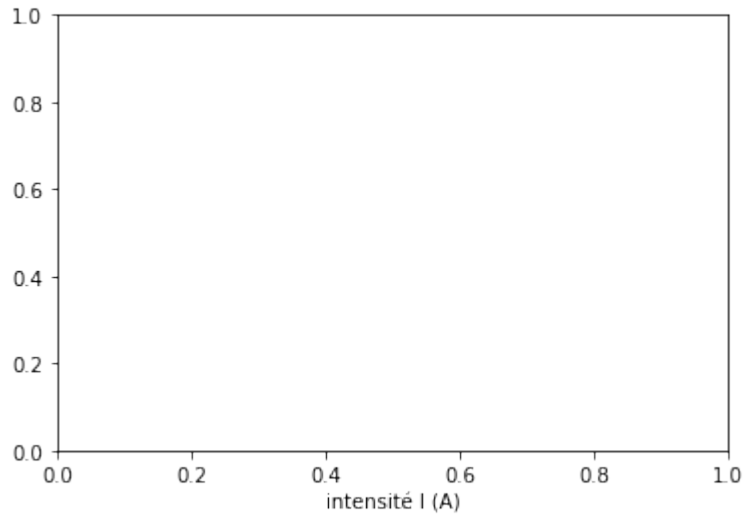
[8]:
```
<matplotlib.legend.Legend at 0x7f309cf55860>
```




commentaire :

[9]:
```
plt.xlabel("intensité I (A)")
```

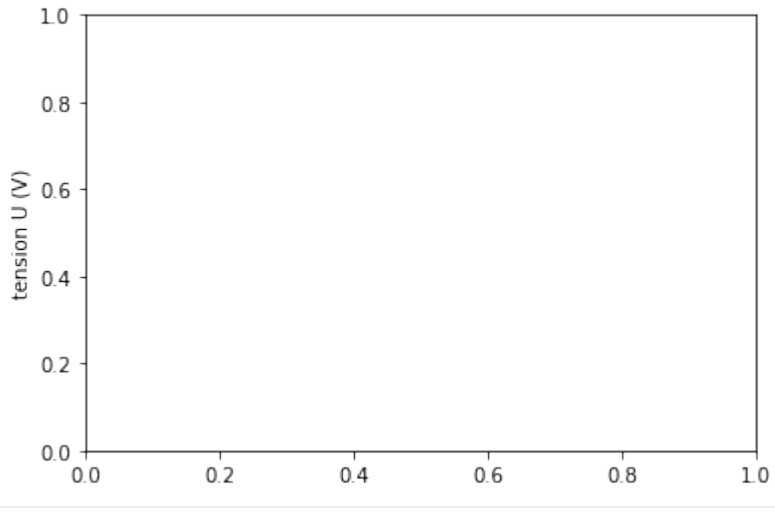
[9]:
```
Text(0.5, 0, 'intensité I (A)')
```




commentaire :

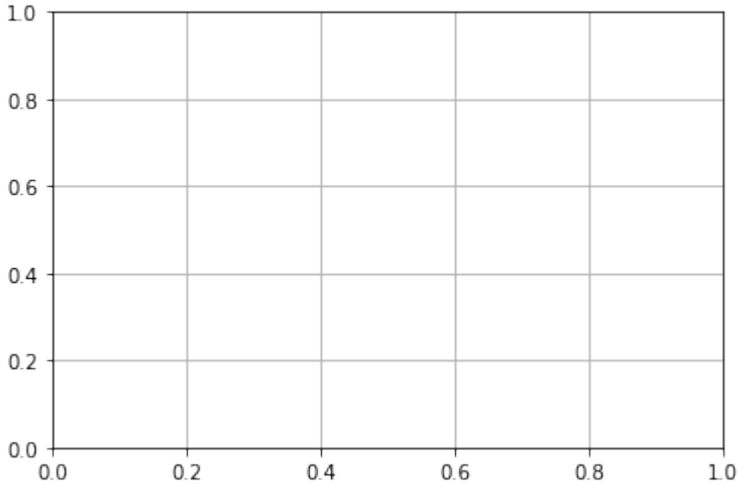
[10]:
```
plt.ylabel("tension U (V)")
```

[10]:
```
Text(0, 0.5, 'tension U (V)')
```




commentaire :

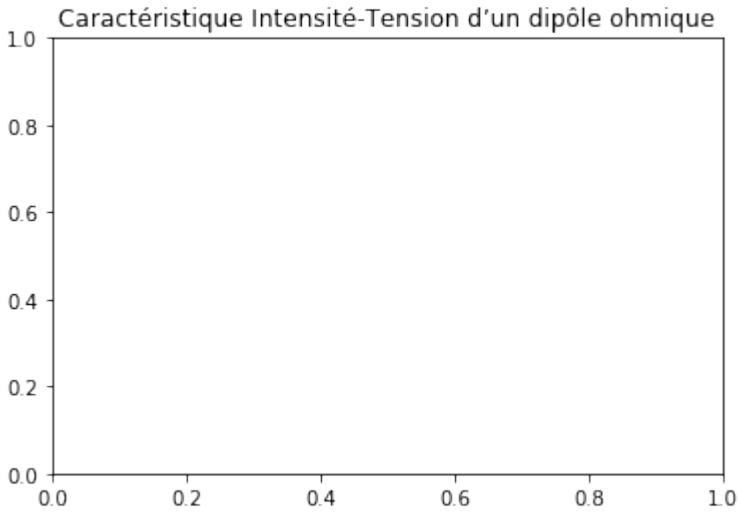
[11]:
```
plt.grid()
```




commentaire :

[12]:
```
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
```

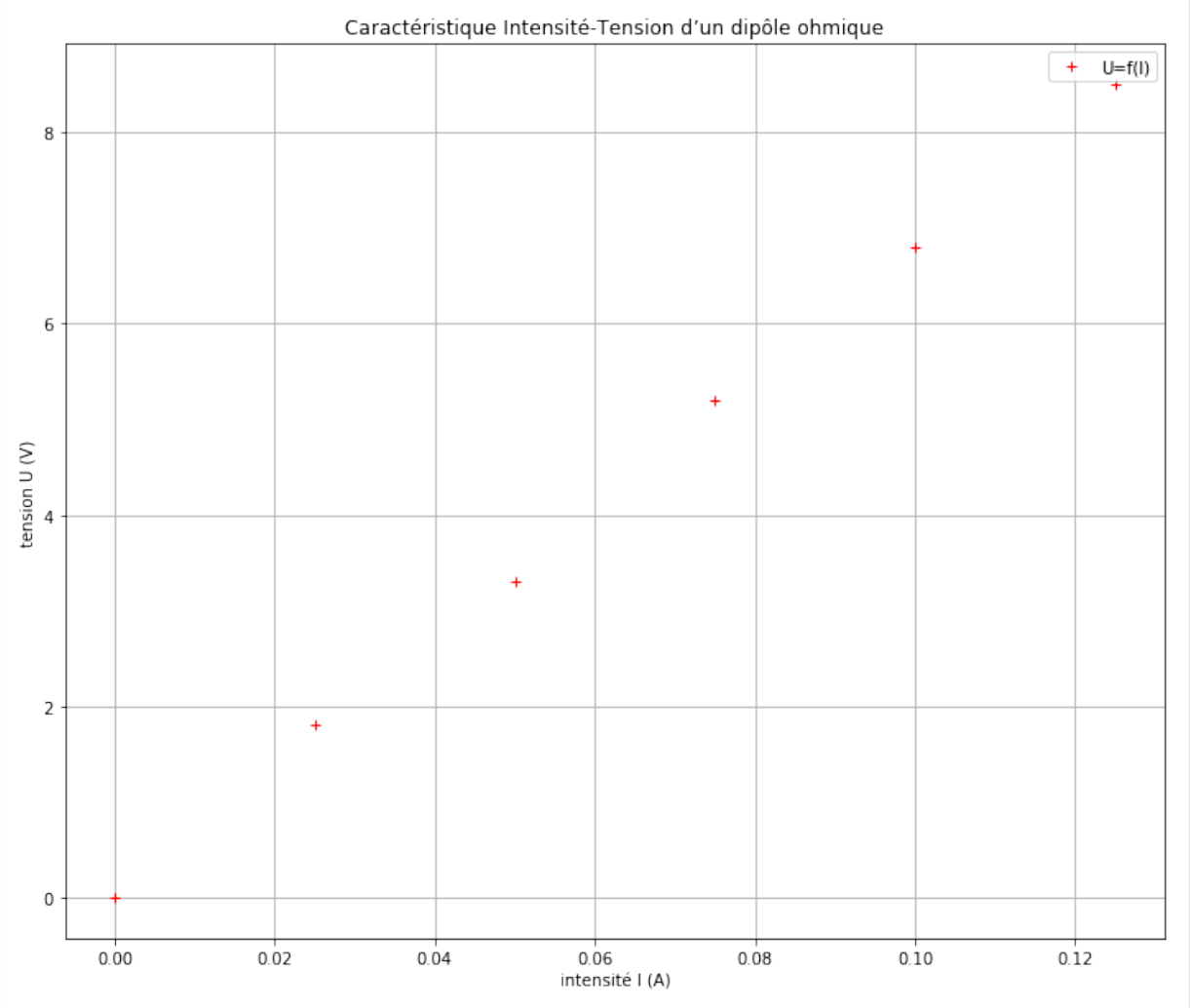
[12]:
```
Text(0.5, 1.0, 'Caractéristique Intensité-Tension d'un dipôle ohmique')
```




commentaire :

4. Exécutez maintenant le programme en entier !

[13]:
```
import numpy as np
import matplotlib.pyplot as plt
I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
U=np.array([0,1.8,3.3,5.2,6.8,8.5])
fig = plt.figure(figsize=(12,10))
plt.plot(I,U,'r+',label='U=f(I)')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
plt.show()
```




5. Il s'agit maintenant de modéliser la courbe obtenue.

5.1. Quelle est la forme de la courbe obtenue ?

5.2. Quelle est l'équation mathématique d'une telle courbe ?

5.3. Exécutez le programme ci-dessous permettant de modéliser la courbe obtenue par une droite.

[14]:
```
from scipy import stats
slope, intercept, r_value, p_value, std_error = stats.linregress(I, U)
print ('slope {0:.2f}'.format(slope))
print('intercept {0:.2f}'.format(intercept))
Umodel = slope*I+intercept
print('U= {0:.2f}'.format(slope),'x I')
print ('Le coefficient de corrélation r vaut {0:.4f}'.format(r_value))
print('Les valeurs de la tension modélisée sont',Umodel)
```

```
slope 67.89
intercept 0.02
U= 67.89 x I
Le coefficient de corrélation r vaut 0.9997
Les valeurs de la tension modélisée sont [0.02380952 1.72095238 3.41809524 5.
↪1152381  6.81238095 8.50952381]
```

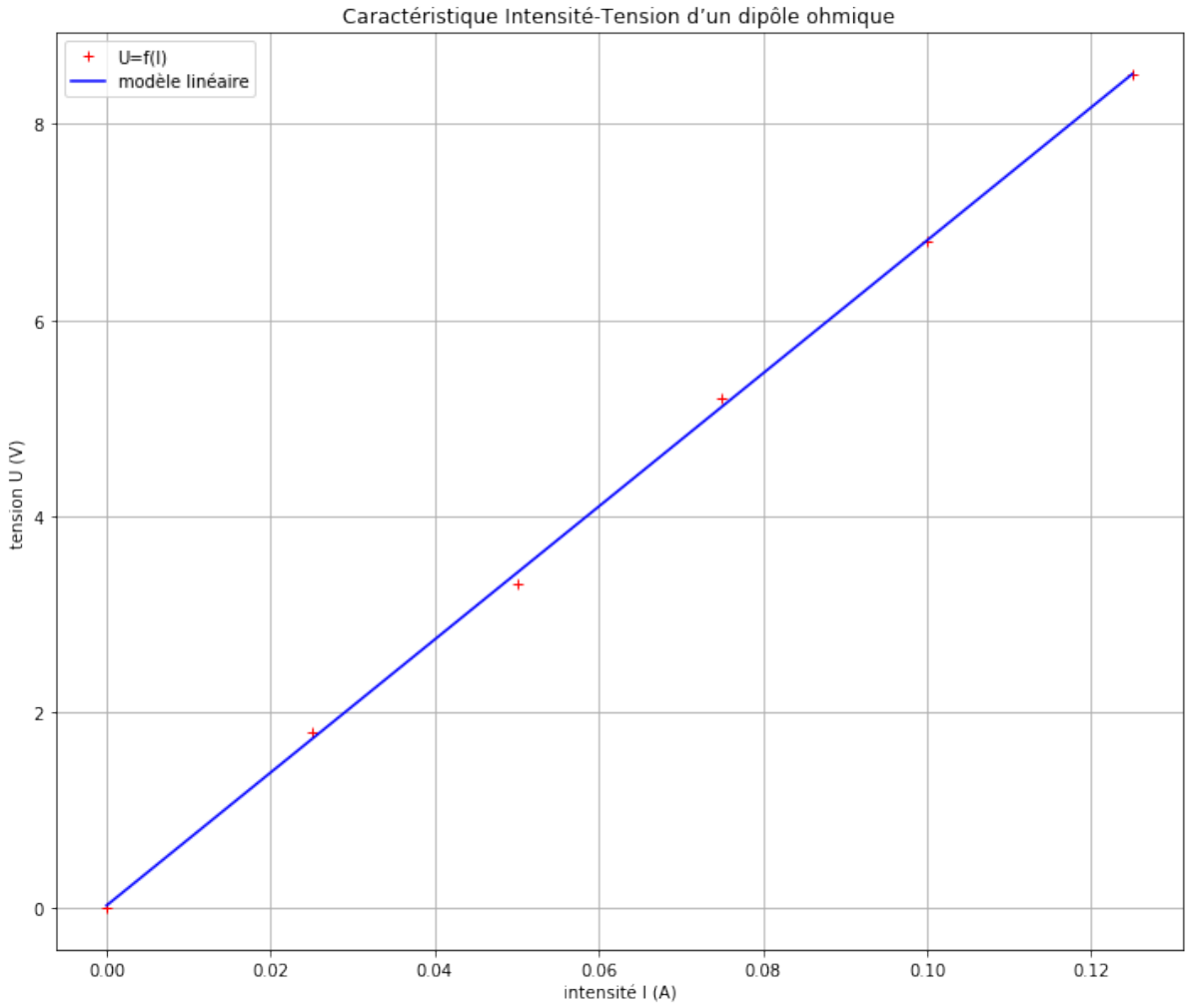
5.3.1. Que représente l'objet slope ?

5.3.2. Que représente l'objet intercept ?

5.3.3. Que représente l'objet r_value ?

5.4. Affichez la droite modélisée grâce au programme ci-dessous.

[15]:
```
fig = plt.figure(figsize=(12,10))
plt.plot(I,U,'r+',label='U=f(I)')
plt.plot(I,Umodel,'b',label='modèle linéaire')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
plt.show()
```




5.4.1. La tension U et l'intensité I sont-elles proportionnelles ? Pourquoi ?

5.4.2. Que remarquez-vous à propos de la valeur du coefficient directeur de la droite ?

5.4.3. En déduire une formule appelée loi d'Ohm entre la tension U, l'intensité I et la résistance électrique R du dipôle ohmique.

### 2.5.8 La loi d'Ohm (version polyfit et avec fonctions)

Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

| I (mA) | 0 | 25 | 50 | 75 | 100 | 125 |
|---|---|---|---|---|---|---|
| U (V) | 0 | 1,8 | 3,3 | 5,2 | 6,8 | 8,5 |

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# création de la fonction modelisation
# modélisation par une droite d'équation
# y=ax+b (polynôme de degré 1)

# la fonction polyfit permet de déterminer les
# coefficients a=coef[0] et b=coef[1] de la droite
# ymodel permet de déterminer les valeurs modélisées de y

def modelisation(x,y):
    coef=np.polyfit(x,y,1)
    ymodel=coef[0]*x+coef[1]
    print ('U= {0:.1f}'.format(coef[0]),'x I')
    print('Les valeurs de la tension modélisée sont',ymodel)
    return (ymodel)
```

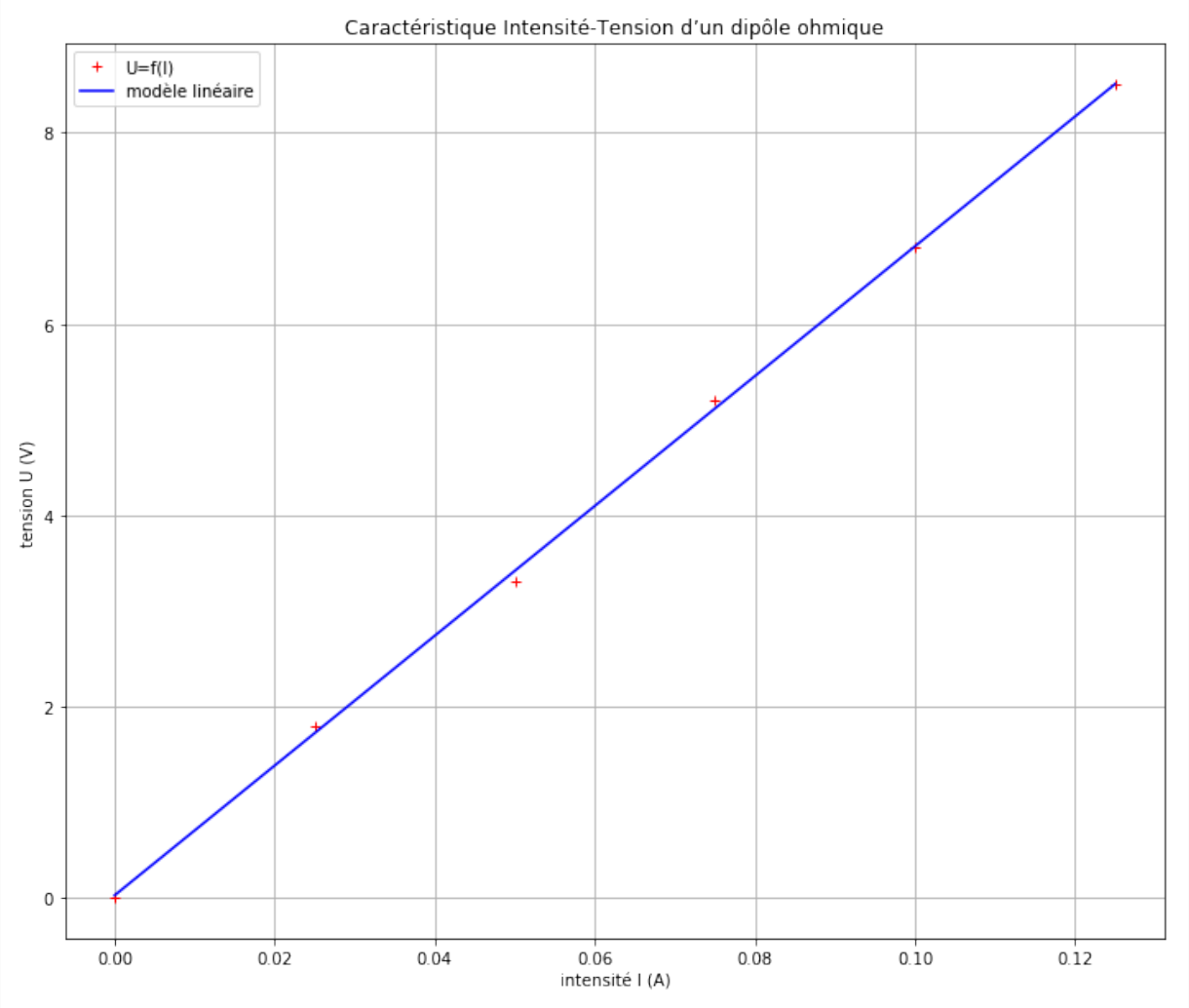
[3]:
```
# fonction permettant de tracer le graphique avec les points
# expérimentaux et la courbe obtenue après modélisation

def courbemodelisee (x,y,ymodel) :
    fig = plt.figure(figsize=(12,10))
    plt.plot(x,y,'r+',label='U=f(I)')
    plt.plot(x,ymodel,'b',label='modèle linéaire')
    plt.legend()
    plt.xlabel("intensité I (A)")
    plt.ylabel("tension U (V)")
    plt.grid()
    plt.title("Caractéristique Intensité-Tension d'un "
              "dipôle ohmique")
    plt.show()
```

[4]:
```
# tableaux numpy obligatoires à cause de l'opération vectorisée
# permettant de créer Umodel

I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
U=np.array([0,1.8,3.3,5.2,6.8,8.5])
Umodel=modelisation(I,U)
courbemodelisee(I,U,Umodel)
```

```
U= 67.9 x I
Les valeurs de la tension modélisée sont [0.02380952 1.72095238 3.41809524 5.
↪1152381  6.81238095 8.50952381]
```




### 2.5.9 La loi d'Ohm (version professeur, polyfit et sans fonction)

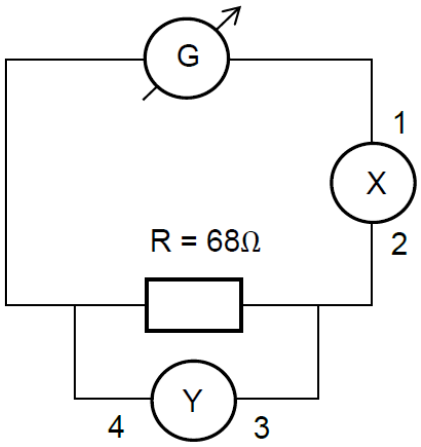
Mathilde, élève de 2nde, souhaite tracer la caractéristique d'un dipôle ohmique, c'est-à-dire la courbe donnant les valeurs de la tension aux bornes du dipôle ohmique en fonction des valeurs de l'intensité du courant qui le traverse.

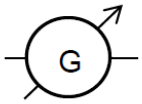
`Télécharger le pdf`

`Télécharger le notebook`

`Lancer le notebook sur binder (lent)`

Elle a schématisé le circuit de son expérience :




Symbole d'un générateur dont on peut faire varier la tension.

1. Dans la cellule ci-dessous, indiquer la signification des symboles X et Y et le nom des bornes 1, 2, 3, 4.

X : 1 : 2 : Y : 3 : 4 :

Mathilde relève les mesures expérimentales suivantes :

| I (mA) | 0 | 25 | 50 | 75 | 100 | 125 |
|---|---|---|---|---|---|---|
| U (V) | 0 | 1,8 | 3,3 | 5,2 | 6,8 | 8,5 |

2. Aider Mathilde à coder la deuxième ligne du tableau de valeurs dans la cellule vide ci-dessous en vous aidant du code de la première ligne (attention les valeurs de l'intensité y ont été converties en ampère).

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
 # array signifie tableau en anglais
I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
print (I)
```

```
[0.    0.025 0.05  0.075 0.1   0.125]
```

[3]:
```
U=np.array([0,1.8,3.3,5.2,6.8,8.5])
print(U)
```

```
[0.  1.8 3.3 5.2 6.8 8.5]
```

3. Mathilde veut maintenant afficher la caractéristique « intensité-tension » du dipôle ohmique en respectant les consignes suivantes :
   — axe des abscisses (horizontal) : Intensité I (mA)
   — axe des ordonnées (vertical) : Tension U(V)
   — points expérimentaux : croix + de couleur rouge
   — Titre : « Caractéristique Intensité-Tension d'un dipôle ohmique »

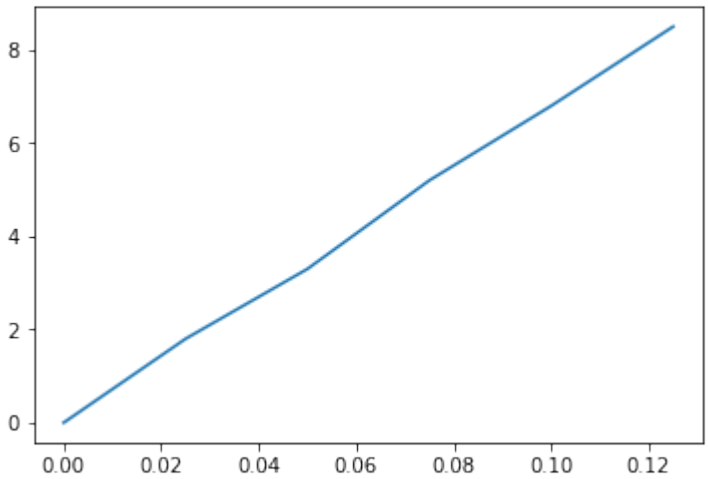
Les cellules ci-dessous contiennent chacune une ligne du code nécessaire à l'affichage de la caractéristique. Exécuter chaque cellule au fur et à mesure afin de comprendre leur utilité. Noter si besoin des commentaires dans les cellules laissées vides à cet effet.

[4]:
```
fig = plt.figure(figsize=(12,10))
```

```
<Figure size 864x720 with 0 Axes>
```

[5]:
```
plt.plot(I,U)
```

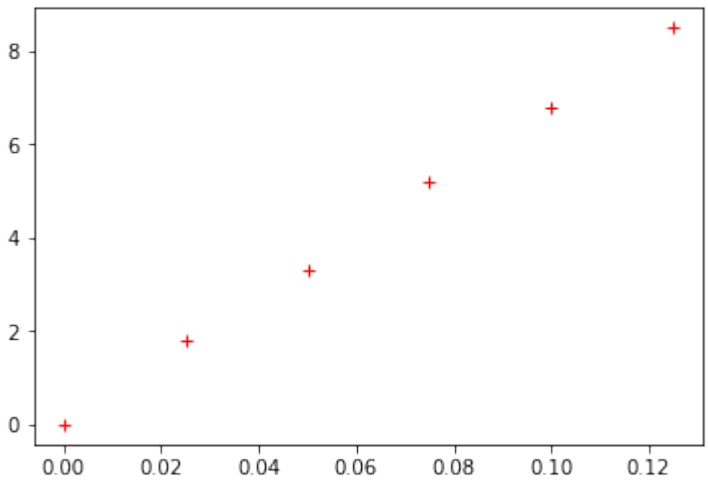
[5]:
```
[<matplotlib.lines.Line2D at 0x7f6500207be0>]
```



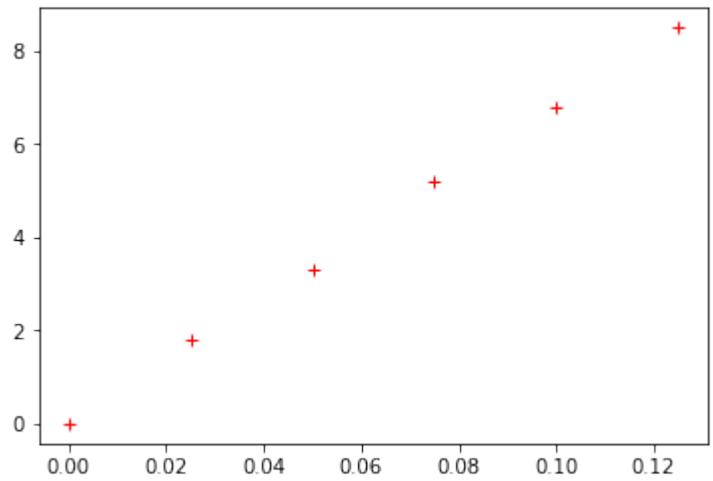

commentaire :

[6]:
```
plt.plot(I,U,'r+')
```

[6]:
```
[<matplotlib.lines.Line2D at 0x7f65001992b0>]
```



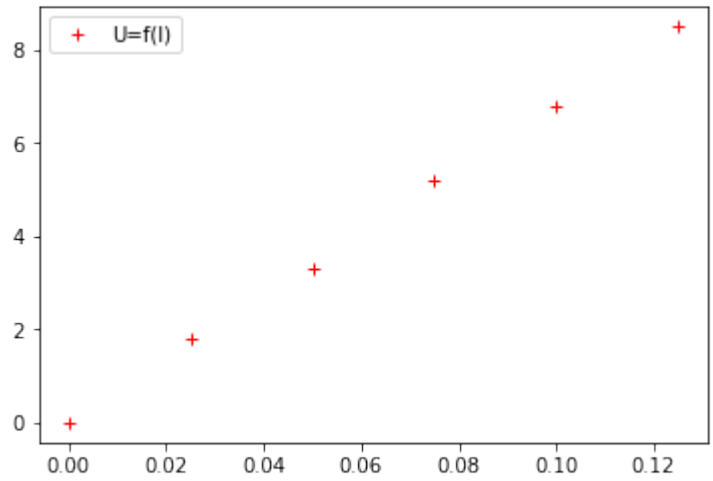

[7]:
```
plt.plot(I,U,'r+')
plt.show()
```




commentaire :

[8]:
```
plt.plot(I,U,'r+',label='U=f(I)')
plt.legend()
```

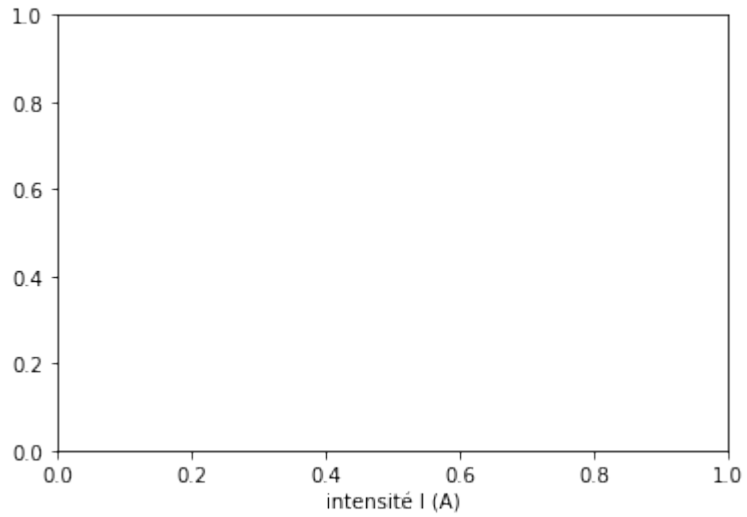
[8]:
```
<matplotlib.legend.Legend at 0x7f6500135898>
```




commentaire :

[9]:
```
plt.xlabel("intensité I (A)")
```

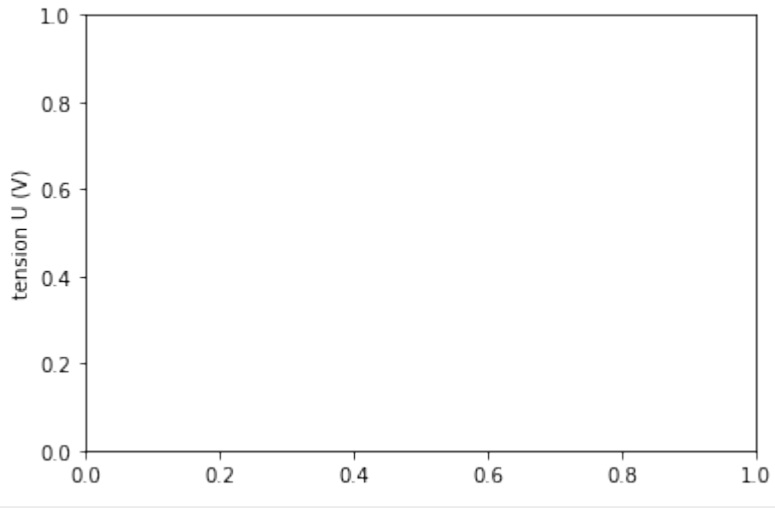
[9]:
```
Text(0.5, 0, 'intensité I (A)')
```




commentaire :

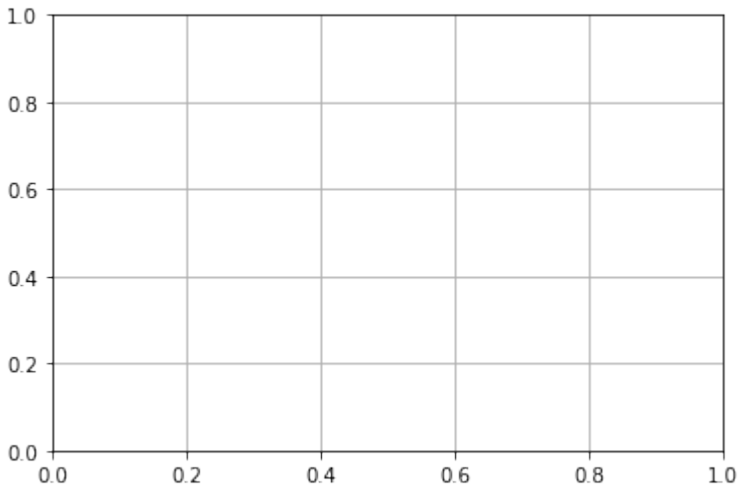
[10]:
```
plt.ylabel("tension U (V)")
```

[10]:
```
Text(0, 0.5, 'tension U (V)')
```




commentaire :

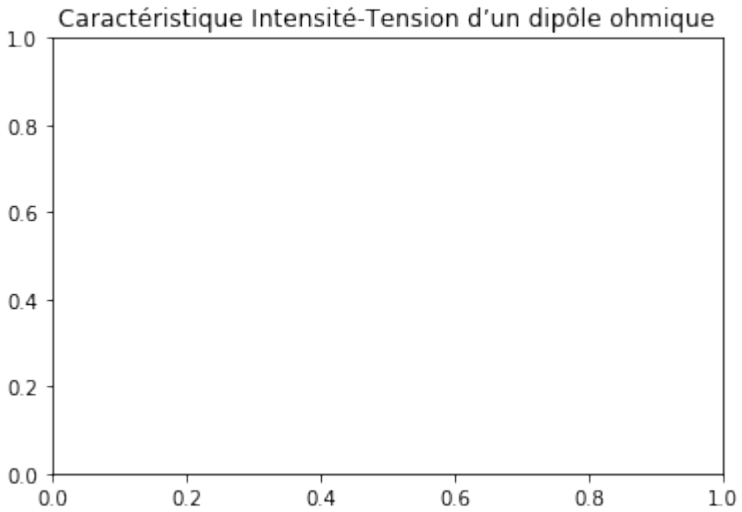
[11]:
```
plt.grid()
```




commentaire :

[12]:
```
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
```

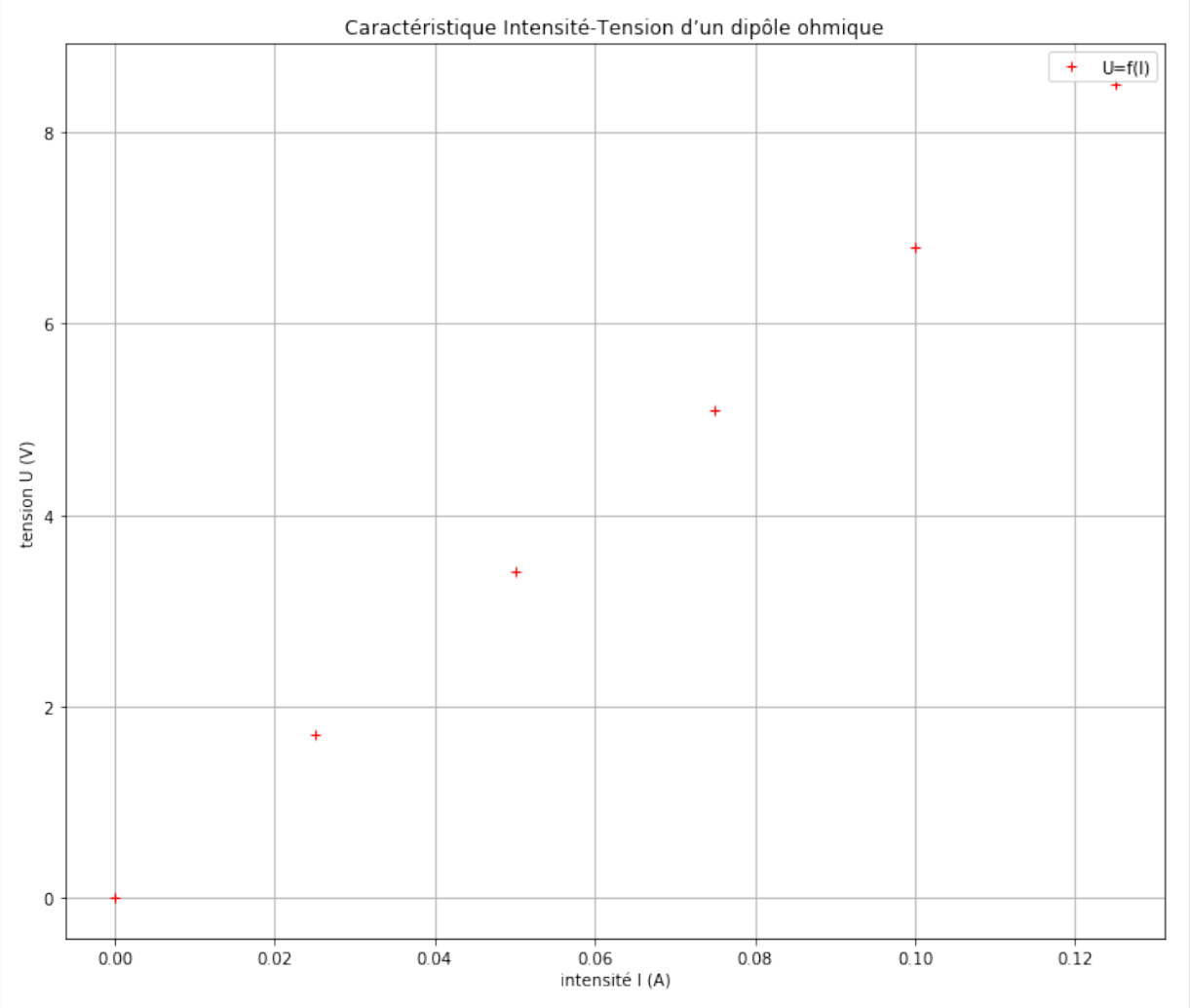
[12]:
```
Text(0.5, 1.0, 'Caractéristique Intensité-Tension d'un dipôle ohmique')
```




commentaire :

4. Exécutez maintenant le programme en entier !

[13]:
```
import numpy as np
import matplotlib.pyplot as plt
I=np.array([0,25e-3,50e-3,75e-3,100e-3,125e-3])
U=np.array([0,1.7,3.4,5.1,6.8,8.5])
fig = plt.figure(figsize=(12,10))
plt.plot(I,U,'r+',label='U=f(I)')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
plt.show()
```




5. Il s'agit maintenant de modéliser la courbe obtenue.

5.1. Quelle est la forme de la courbe obtenue ?

5.2. Quelle est l'équation mathématique d'une telle courbe ?

5.3. Exécutez le programme ci-dessous permettant de modéliser la courbe obtenue par une droite.

[14]:
```
coeff=np.polyfit(I, U,1)
print ('{0:.1f}'.format(coeff[0]),
       '{0:.1f}'.format(coeff[1]))
Umodel = coeff[0]*I+coeff[1]
print('U={0:.1f}'.format(coeff[0]),'x I')
print('Les valeurs de la tension modélisée sont',Umodel)
```

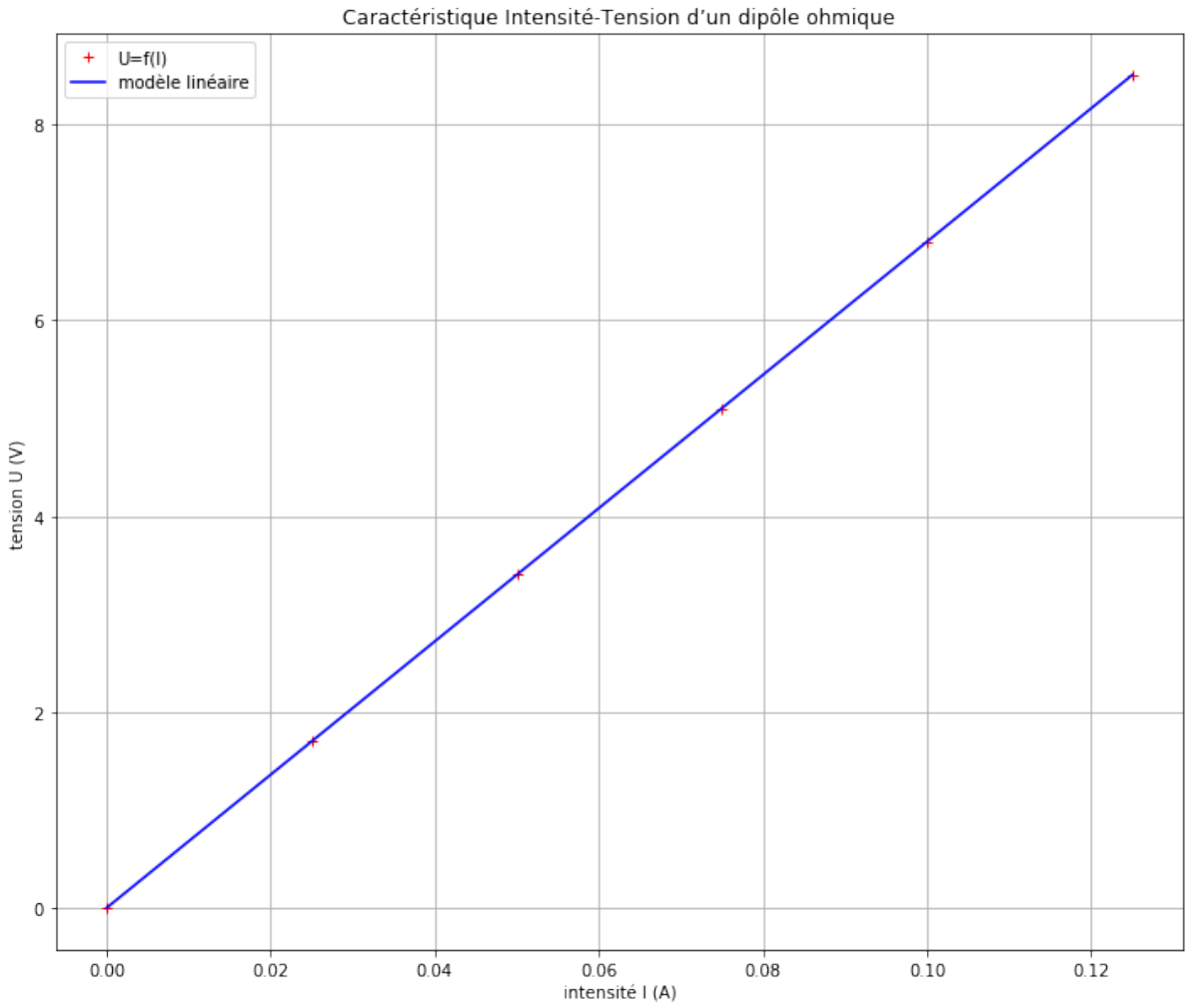
```
68.0 -0.0
U=68.0 x I
Les valeurs de la tension modélisée sont [-1.59473833e-16  1.70000000e+00  3.
↪40000000e+00  5.10000000e+00
  6.80000000e+00  8.50000000e+00]
```

5.3.1. Que représente coeff[0] ?

5.3.2. Que représente coeff[1] ?

5.4. Affichez la droite modélisée grâce au programme ci-dessous.

[15]:
```
fig = plt.figure(figsize=(12,10))
plt.plot(I,U,'r+',label='U=f(I)')
plt.plot(I,Umodel,'b',label='modèle linéaire')
plt.legend()
plt.xlabel("intensité I (A)")
plt.ylabel("tension U (V)")
plt.grid()
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
plt.show()
```




5.4.1. La tension U et l'intensité I sont-elles proportionnelles ? Pourquoi ?

5.4.2. Que remarquez-vous à propos de la valeur du coefficient directeur de la droite ?

5.4.3. En déduire une formule appelée loi d'Ohm entre la tension U, l'intensité I et la résistance électrique R du dipôle ohmique.

### 2.5.10 Tracé des vecteurs vitesse (version sans fonction)

`Télécharger le pdf`

`Télécharger le notebook`, `le fichier csv` et `la vidéo`,

`Lancer le notebook sur binder (lent)`

[1]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import csv
```

[2]:
```
with open("chute_balle.csv", 'r', encoding='utf-8') as f :
    rfichier = csv.reader(f, delimiter=";")
    tableau=[]
    index_row=0  # indice d'une ligne du fichier
    N=1 # nombre de lignes d'en-têtes
    for row in rfichier:
        if index_row < N:
            index_row = index_row+1
        else :
            for i in range (len(row)):
                if len(tableau) <= i:
                    X = []
                    tableau.append(X)
                try:
                    tableau[i].append(float(row[i].replace(",",'.')))
                except ValueError:
                    print('erreur:contenu de cellule non numérique')
                    continue
t=tableau[0]
print(t)
x=tableau[1]
print(x)
y=tableau[2]
print(y)
```

```
[0.76, 0.8, 0.84, 0.88, 0.92, 0.96, 1.0, 1.04, 1.08, 1.12, 1.16, 1.2, 1.24, 1.28,␣
↪1.32, 1.36]
[0.00865710739046, 0.106770991149, 0.204884874908, 0.302998758666, 0.398226939961,␣
↪0.493455121256, 0.588683302551, 0.675254376456, 0.767596855287, 0.857053631656,␣
↪0.946510408024, 1.03019577946, 1.11676685337, 1.19756652235, 1.27836619132, 1.
↪36205156277]
[1.98247759242, 1.97093478256, 1.94496346039, 1.89590651851, 1.83530676678, 1.
↪7545070978, 1.66216461897, 1.55827933028, 1.43707982682, 1.29568040611, 1.
↪15139528293, 0.984024540049, 0.807996689776, 0.605997517332, 0.403998344888, 0.
↪187570660127]
```

[3]:
```
vx=[]
for i in range (len(x)-1):
    vxi=(x[i+1]-x[i])/(t[i+1]-t[i])
    vx.append(vxi)
```

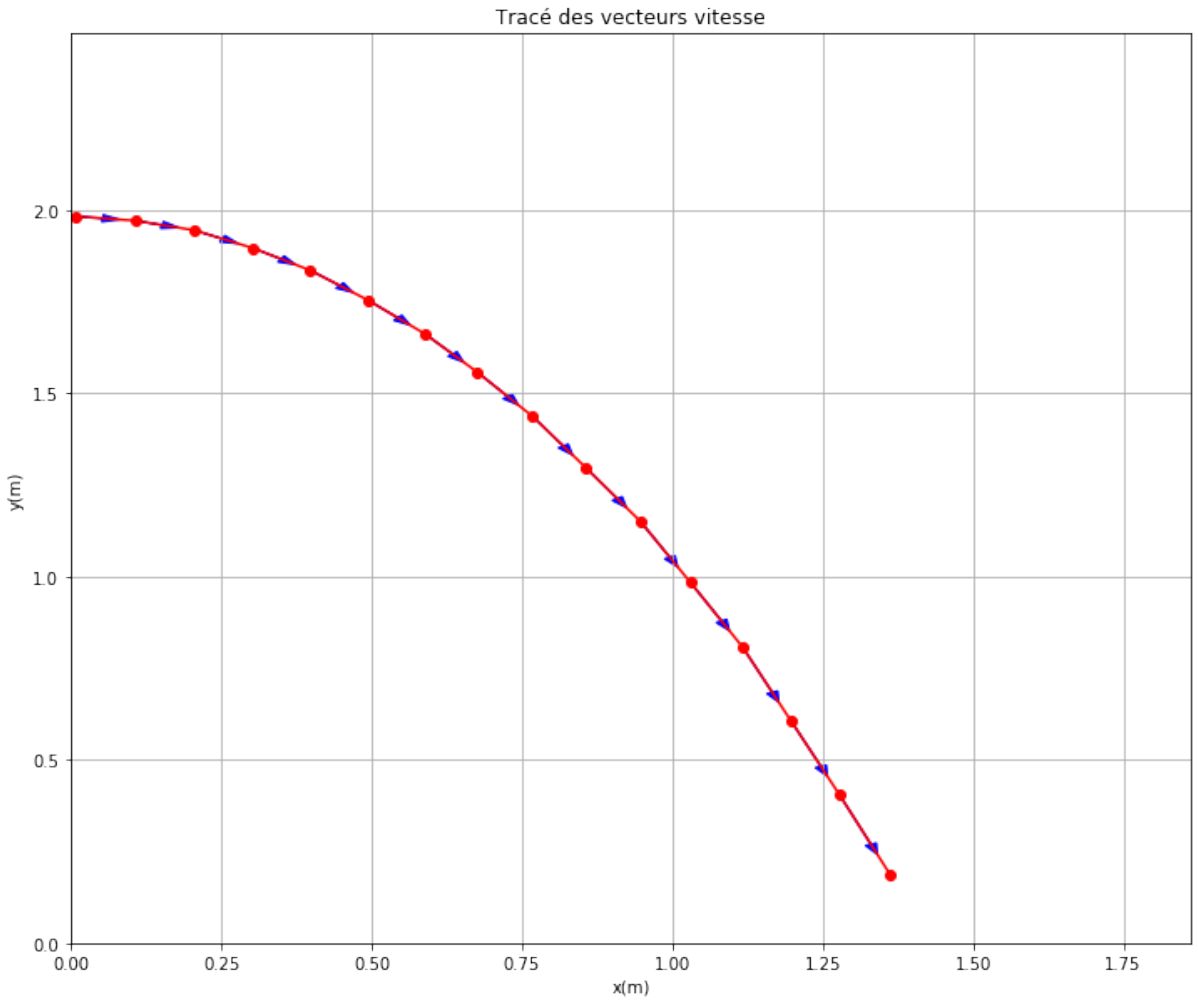
[4]:
```
vy=[]
for i in range (len(y)-1):
    vyi=(y[i+1]-y[i])/(t[i+1]-t[i])
    vy.append(vyi)
```

[5]:
```
fig = plt.figure(1,figsize=(12,10))
plt.plot(x,y,'ro-')
plt.xlim(0,max(x)+0.5)
plt.ylim(0,max(y)+0.5)
plt.grid()
plt.xlabel("x(m)")
plt.ylabel("y(m)")
for i in range (len (vx)) :
    plt.arrow(x[i],y[i],0.03*vx[i],0.03*vy[i],
              fc='b',ec='b',head_width=0.02,
              length_includes_head=True)
```

(continues on next page)

(suite de la page précédente)

```
plt.title("Tracé des vecteurs vitesse")
plt.show()
```




[ ]:

### 2.5.11 Tracé des vecteurs vitesse (version avec fonctions)

Télécharger le pdf

Télécharger le notebook, le fichier csv et la vidéo,

Lancer le notebook sur binder (lent)

[1]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import csv
```

[2]:
```
def charge_fichier_csv(fichier, delimiter=";",N=0):
    with open(fichier, 'r', encoding='utf-8') as f :
        rfichier = csv.reader(f, delimiter=delimiter)
        tableau=[]
        index_row=0
        for row in rfichier:
            if index_row < N:
```

(continues on next page)

(suite de la page précédente)

```
                index_row = index_row+1
            else :
                for i in range (len(row)):
                    if len(tableau) <= i:
                        X = []
                        tableau.append(X)
                    try:
                        tableau[i].append(float(row[i].replace(",",'.')))
                    except ValueError:
                        print('erreur:contenu de cellule non numérique')
                        continue

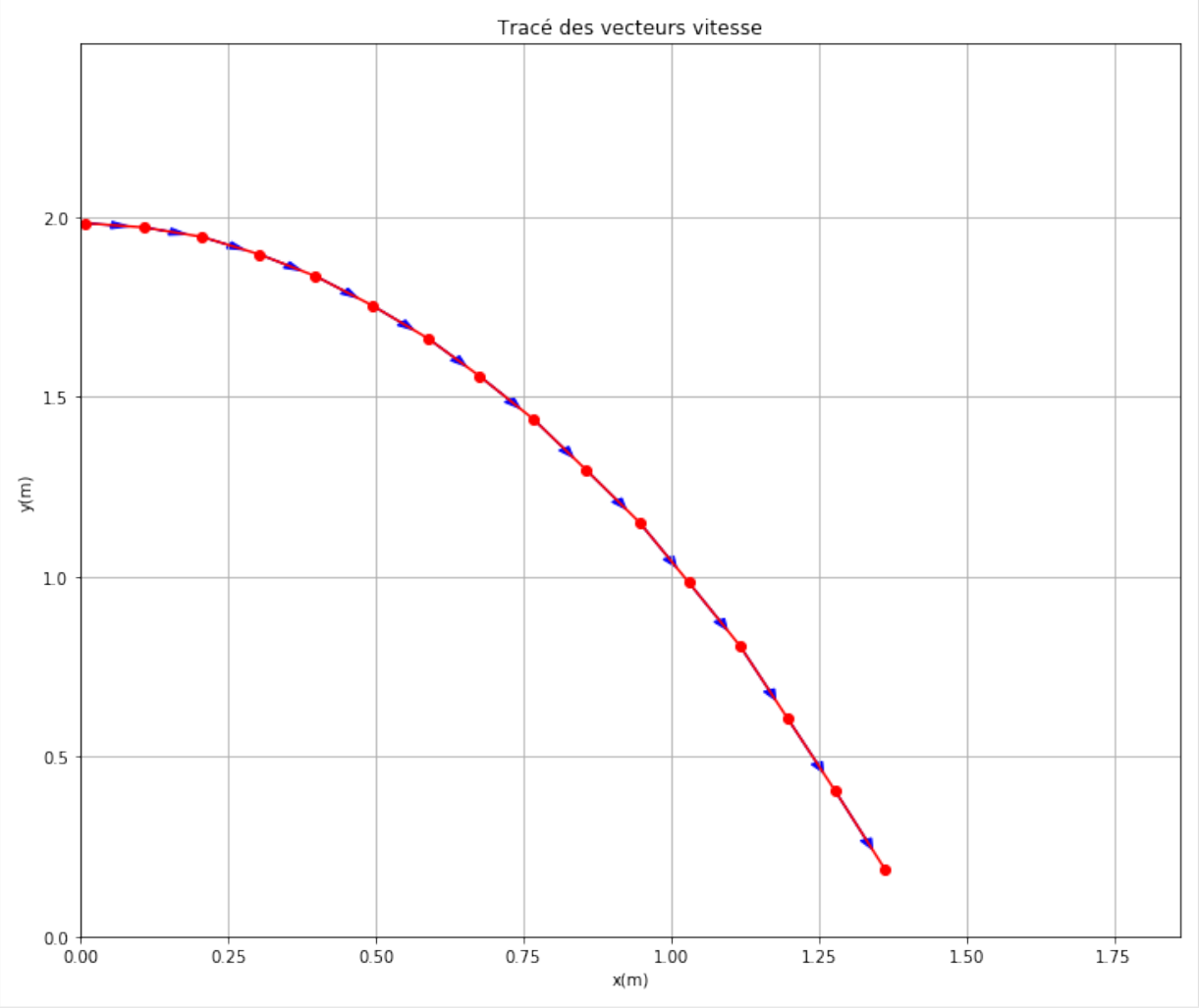
        return (tableau)
```

[3]:
```
def vitesse(t,z):
    vz=[]
    for i in range (len(z)-1):
        vzi=(z[i+1]-z[i])/(t[i+1]-t[i])
        vz.append(vzi)
    return vz
```

[4]:
```
def graphvect(x,y,vx,vy):
    fig = plt.figure(1,figsize=(12,10))
    plt.plot(x,y,'ro-')
    plt.xlim(0,max(x)+0.5)
    plt.ylim(0,max(y)+0.5)
    plt.grid()
    plt.xlabel("x(m)")
    plt.ylabel("y(m)")
    for i in range (len (vx)) :
        plt.arrow(x[i],y[i],0.03*vx[i],0.03*vy[i],fc='b',
                  ec='b',head_width=0.02,
                  length_includes_head=True)
    plt.title("Tracé des vecteurs vitesse")
    plt.show()
```

[6]:
```
tableau = charge_fichier_csv("chute_balle.csv",N=1)
t=tableau[0]
print(t)
x=tableau[1]
print(x)
y=tableau[2]
print(y)
vx=vitesse(t,x)
vy=vitesse(t,y)
graphvect(x,y,vx,vy)
```

```
[0.76, 0.8, 0.84, 0.88, 0.92, 0.96, 1.0, 1.04, 1.08, 1.12, 1.16, 1.2, 1.24, 1.28,␣
↪1.32, 1.36]
[0.00865710739046, 0.106770991149, 0.204884874908, 0.302998758666, 0.398226939961,␣
↪0.493455121256, 0.588683302551, 0.675254376456, 0.767596855287, 0.857053631656,␣
↪0.946510408024, 1.03019577946, 1.11676685337, 1.19756652235, 1.27836619132, 1.
↪36205156277]
[1.98247759242, 1.97093478256, 1.94496346039, 1.89590651851, 1.83530676678, 1.
↪7545070978, 1.66216461897, 1.55827933028, 1.43707982682, 1.29568040611, 1.
↪15139528293, 0.984024540049, 0.807996689776, 0.605997517332, 0.403998344888, 0.
↪187570660127]
```




[ ]:

### 2.5.12 Calcul de la structure d'un atome

Télécharger le pdf

Télécharger le notebook

Lancer le notebook sur binder (lent)

[1]:
```
# programme calculant la structure de l'atome

#données
# la masse des protons et des neutrons est approchée:
masse_nucleon = 1.67e-27
# masse de l'électron
masse_electron = 9.109e-31
# liste des orbitales atomiques
liste_orbitales = ( (1,'s',2), (2,'s',2), (2, 'p', 6), (3, 's', 2), (3, 'p', 6),
                    (4, 's', 2), (3, 'd', 10), (4, 'p', 6), (5, 's', 2), (4, 'd',␣
↪10), (5, 'p', 6),
                    (6, 's', 2), (4, 'f', 14), (5, 'd', 10), (6, 'p', 6),
                    (7, 's', 2), (5, 'f', 14), (6, 'd', 10), (7, 'p', 6))
# liste des éléments chimiques de la classification
elements = (
```

(continues on next page)

(suite de la page précédente)

```
"Hydrogène H",
"Hélium He",
"Lithium Li",
"Béryllium Be",
"Bore B",
"Carbone C",
"Azote N",
"Oxygène O",
"Fluor F",
"Néon Ne",
"Sodium Na",
"Magnésium Mg",
"Aluminium Al",
"Silicium Si",
"Phosphore P",
"Soufre S",
"Chlore Cl",
"Argon Ar",
"Potassium K",
"Calcium Ca",
"Scandium Sc",
"Titane Ti",
"Vanadium V",
"Chrome Cr",
"Manganèse Mn",
"Fer Fe",
"Cobalt Co",
"Nickel Ni",
"Cuivre Cu",
"Zinc Zn",
"Gallium Ga",
"Germanium Ge",
"Arsenic As",
"Sélénium Se",
"Brome Br",
"Krypton Kr",
"Rubidium Rb",
"Strontium Sr",
"Yttrium Y",
"Zirconium Zr",
"Niobium Nb",
"Molybdène Mo",
"Technétium Tc",
"Ruthénium Ru",
"Rhodium Rh",
"Palladium Pd",
"Argent Ag",
"Cadmium Cd",
"Indium In",
"Étain Sn",
"Antimoine Sb",
"Tellure Te",
"Iode I",
"Xénon Xe",
"Césium Cs",
"Baryum Ba",
"Lanthane La",
"Cérium Ce",
"Praséodyme Pr",
"Néodyme Nd",
"Prométhium Pm",
```

(continues on next page)

(suite de la page précédente)

```
"Samarium Sm",
"Europium Eu",
"Gadolinium Gd",
"Terbium Tb",
"Dysprosium Dy",
"Holmium Ho",
"Erbium Er",
"Thulium Tm",
"Ytterbium Yb",
"Lutécium Lu",
"Hafnium Hf",
"Tantale Ta",
"Tungstène W",
"Rhénium Re",
"Osmium Os",
"Iridium Ir",
"Platine Pt",
"Or Au",
"Mercure Hg",
"Thallium Tl",
"Plomb Pb",
"Bismuth Bi",
"Polonium Po",
"Astate At",
"Radon Rn",
"Francium Fr",
"Radium Ra",
"Actinium Ac",
"Thorium Th",
"Protactinium Pa",
"Uranium U",
"Neptunium Np",
"Plutonium Pu",
"Américium Am",
"Curium Cm",
"Berkélium Bk",
"Californium Cf",
"Einsteinium Es",
"Fermium Fm",
"Mendélévium Md",
"Nobélium No",
"Lawrencium Lr",
"Rutherfordium Rf",
"Dubnium Db",
"Seaborgium Sg",
"Bohrium Bh",
"Hassium Hs",
"Meitnérium Mt",
"Darmstadtium Ds",
"Roentgenium Rg",
"Ununbium Uub",
"Ununtrium Uut",
"Ununquadium Uuq",
"Ununpentium Uup",
"Ununhexium Uuh",
"Ununseptium Uus",
"Ununoctium Uuo")

# Entrées
# demande des numéros atomiques et nombre de masse
Z = int(input('Entrer le numéro atomique Z =  '))
```

(continues on next page)

(suite de la page précédente)

```
A = int(input('Entrer le nombre de masse A =  '))

#calcul des masses
masse_noyau = A * masse_nucleon
masse = A * masse_nucleon + Z * masse_electron

# calcul des couches électroniques

if Z <= 2:
    couches = "(K)" + str(Z)
elif Z <=10:
    couches = "(K)" + str(2) + "(L)" + str(Z-2)
elif Z <= 18:
    couches = "(K)" + str(2) + "(L)" + str(8) + "(M)" + str(Z-10)
else:
    couches = "ce calcul est limité à des numéros atomiques inférieurs ou égal à 18
↪"


# calcul des orbitales atomiques
orbitale = 0
n_restant = Z
structure = ""
while n_restant > 0:
    (n, nom, ne) = liste_orbitales [orbitale]
    if n_restant < ne:
        nmin = n_restant
    else:
        nmin = ne
    structure = structure + str(n) + nom + str(nmin) + ' '
    n_restant = n_restant - nmin
    orbitale = orbitale + 1

# impression des résultats
print ("\nRESULTATS")
# impression de l'élément et de son symbole
print('{:35}'.format("il s'agit de l'élément "), elements[Z-1])
# impression de la structure de l'atome
print('{:35}'.format('le nombre de protons est: '), Z )
print('{:35}'.format('le nombre de neutrons est: '), A - Z)
print('{:35}'.format("le nombre d'électrons est: "), Z)
# impression des masses, du noyau et de l'atome
print("{0:35} {1:.3e} {2:8}".format("la masse du noyau de l'atome est: ",masse_
↪noyau,' kg'))
print('{:35}'.format("la masse de l'atome est: "), "{0:.3e}".format(masse), ' kg')
print("")
# impression des couches et structure électronique
print('{:65}'.format("le remplissage des couches électroniques donne: "), couches)
print('{:65}'.format('Selon la règle de Klechkowski, la structure électronique␣
↪est: '), structure)
```

```
Entrer le numéro atomique Z =  6
Entrer le nombre de masse A =  14

RESULTATS
il s'agit de l'élément              Carbone C
le nombre de protons est:           6
le nombre de neutrons est:          8
le nombre d'électrons est:          6
la masse du noyau de l'atome est:   2.338e-26  kg
la masse de l'atome est:            2.339e-26  kg
```

(continues on next page)

(suite de la page précédente)

```
le remplissage des couches électroniques donne:                    (K)2(L)4
Selon la règle de Klechkowski, la structure électronique est:      1s2 2s2 2p2
```

CHAPITRE 3

# Accès et téléchargements

Le contenu de ce guide est disponible :

- en ligne via ReadTheDocs
- dans une archive ZIP à télécharger
- au format PDF à télécharger
- au format Epub à télécharger
- en mode démo exécutable via binder (le chargement initial peut être long)
- dans l'ENT de l'académie de Rouen, dans le groupe « Python pour la SPC » (en cours de déploiement)
- sur gitlab



**Auteurs, par ordre alphabétique**

- BARBIER Jean-Matthieu <jean-matthieu.barbier@ac-rouen.fr>
- DELACOUR Pascal <pascal.delacour@ac-rouen.fr>
- DENDIEVEL Alexis <alexis.dendievel@ac-rouen.fr>
- DEVEDEUX Dominique <dominique.devedeux@ac-rouen.fr>
- GRANDPRE Caroline <caroline.grandpre@ac-rouen.fr>
- REBOLINI Gaelle <gaelle-nathalie.rebolini@ac-rouen.fr>