Python pour la SPC au lycée - Formation

Version 1.0.1

GP Python

févr. 05, 2020

Table des matières

CHAPITRE 1

Déroulement

Télécharger le `document PDF correspondant`

- Présentation des objectifs de la formation - `intro rapide`.
- Présentation de l'outil **notebook jupyter** et introduction rapide au python
- Mise en activité « multi-entrées » :
  - progression différenciée proposée ci-dessous
  - autonomie à l'aide du site https://pyspc.readthedocs.io
  - autonomie totale - réalisation de programmes en lien avec les nouveaux programmes
- Petite pratique « à la maison » entre les deux sessions

CHAPITRE 2

Progression

2.1 Quelques bases de python pour la SPC

`Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)`

2.1.1 L'environnement

Utilisation des notebook Jupyter
- ENT
- anaconda

Notebook = « mélange » de code et de texte

2.1.2 Le markdown

Le markdown est le langage utilisé pour mettre en forme le texte dans les cellules « markdown » des notebooks. C'est un langage très utilisé dans le monde du développement, parce qu'il permet de mettre en forme du texte tout en conservant une bonne lisibilité.

```
# Titre
## Sous titre
### Sous-sous titre
#### ....

On peut formater 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
```

(continues on next page)

(suite de la page précédente)

```
3. liste numérotée

Pour changer de paragraphe
il faut
sauter
une ligne

Prochain paragraphe

Ecrire des maths : $x^2$ ou une formule séparée : $$\frac{x^2}{\sqrt{3}}$$
```

2.1.3 Les éléments de base

En Python, tout est objet : les nombres, les chaînes de caractères, les listes, les fonctions... Pour déterminer le type d'un objet : fonction `type()`

A l'intérieur du code, un commentaire débute par #

2.1.3.1 Les nombres

- nombres entiers (**int**) : exemple : 3
- nombres à virgule (**float**) exemple : 1.

[1]:
```
type(3)
```

[1]:
```
int
```

[2]:
```
type(1.)
```

[2]:
```
float
```

Les opérations : classiquement, addition (+), soustraction (-), multiplication (*), division (/) et puissance (**)

[3]:
```
# Opérations
print(2*3)
print(6/4)
print(5**6)
```

```
6
1.5
15625
```

[4]:
```
# Conversions de type
# 3.124 --> int = 3
print(int(3.124))
# int -> float
print(float(3))
```

```
3
3.0
```

[5]:
```
# Mise en forme - notation scientifique
print("{0:.4e} {1:.2e}".format(338.14159, 444848.3234234))
# Mise en forme - un nombre donné de décimales
print("{0:.4f} {1:.2f}".format(338.14159, 444848.3234234))
```

```
3.3814e+02 4.45e+05
338.1416 444848.32
```

2.1.3.2 Les caractères

[6]:
```
# Chaîne de caractères
print('bonjour')
print("bonjour")
```

```
bonjour
bonjour
```

2.1.3.3 Les variables

- on assigne une valeur à une variable avec le signe =. Exemple : `a=12` met la valeur 12 dans la variable `a`
- précision sur les noms de variables :
  - **uniquement des lettres sans accents**, ou des underscores _
  - on peut utiliser des **chiffres** mais **pas au début**
  - **pas de majuscules, juste des minuscules**

[7]:
```
x=3
msg = 'Yo'
nom = "Albert"

print(x)
print (x*(msg+" ")+nom)
```

```
3
Yo Yo Yo Albert
```

[8]:
```
nom = nom + " Raymond"
print(nom)
```

```
Albert Raymond
```

2.1.3.4 Les listes

[9]:
```
l = [1, 'toto',3.2 , "5", 12]
# recupérer un element de  la liste
e = l[2]
print(e)
```

```
3.2
```

Fin cours étapes 1 et 2 : faire applications correspondantes (dissolution et structures des atomes)

2.1.4 Les fonctions

2.1.4.1 Les librairies

Par défaut, le python a très peu de fonctionnalités. Mais il existe une énorme **bibliothèque** de fonctions qu'on peut utiliser pour enrichir les fonctionnalités d'un programme. L'index de la bibilothèque se trouve à l'adresse https://docs.python.org/3/library/index.html

En plus de cette bibliothèque de fonctions « par défaut » (bibliothèque standard), il existe des dizaines de milliers de bibliothèques que l'on peut consulter sur https://pypi.org/

Pour utiliser une fonction mathématique, ou d'autres types de fonctions qui ne sont pas connues « de base », il faut les **importer**.

[10]:
```
import math
math.cos(3.14)
math.sin(0)
```

[10]:
```
0.0
```

[11]:
```
# pour importer une seule fonction :

from math import cos
cos(3.14)
```

[11]:
```
-0.9999987317275395
```

[12]:
```
# Pour tout importer (à éviter)

from math import *
cos(3.14)
```

[12]:
```
-0.9999987317275395
```

2.1.5 Créer ses propres fonctions

Une fonction en maths : $f(x) = 3x + 2$ s'écrit en python :

```
def f(x):
    return 3*x+2
```

- contraintes sur le nom de la fonction : pareil que pour les variables
- début de la définition : **def**
- un ou plusieurs arguments donnés entre parenthèses
- le corps de la fonction doit être indenté (4 espaces)
- la valeur retournée par la fonction est renvoyée par `return`
- **la définition de la fonction se termine par :**

[13]:
```
def f(x):
    return 3*x+2
print(f(3))
print(f(15))
```

```
11
47
```

[14]:
```
# Une fonction peut avoir des arguments par défaut
def affine(x, a, b=0):
    return a*x+b

print(affine(2,5,3))
print(affine(2,5,0))
print(affine(2,5))
```

```
13
10
10
```

[15]:
```
# On peut donner les arguments par nom
print(affine(a=5,x=2,b=3))
```

```
13
```

[16]:
```
# Il y a des variables LOCALES dans la fonction

toto = 12
titi = 15

def pouet(x):
    print("Coucou")
    toto=x+2
    tutu=x-2
    print(toto)
    print(titi) # <--- pas d'erreur, on utilise la valeur globale
    print(tutu)

pouet(2)
print(toto)
# print(tutu) <--- ça plante parce que tutu est local dans pouet
```

```
Coucou
4
15
0
12
```

[17]:
```
# Une fonction est une objet comme les autres, on peut donc
# la mettre dans une variable

def f(x):
    return 3*x+2

g=f
print(g(3))
```

```
11
```

2.1.6 Les structures du langage

2.1.6.1 Les booléens et les tests

2.1.6.1.1 Les booléens et les opérations booléennes

Un booléen : valeur vraie ou fausse (0/1)... en python `True` ou `False`

[18]:
```
type(True)
```

[18]:
```
bool
```

[19]:
```
type(False)
```

[19]:
```
bool
```

[20]:
```
print("vrai et vrai = ", True and True)
print("vrai ou faux = ", True or False)
print("non vrai = ", not True)
```

```
vrai et vrai =  True
vrai ou faux =  True
non vrai =  False
```

2.1.6.1.2 Les tests

Les tests renvoient une valeur booléenne

« est-ce que machin est plus grand que truc ? » : vrai / faux

liste des tests disponibles : - > supérieur - < inférieur - >= supérieur ou égal - <= inférieur ou égal - == égal - != différent - `in` présent dans

[21]:
```
print(3<4)
print(2>=2)
print(3!=2)
print(3==3.)
print(3 in [1, 3, 4])
print(2 in [1, 3, 4])
```

```
True
True
True
True
True
False
```

2.1.6.2 Les structures conditionnelles

Plusieurs manière de faire des actions qui dépendent d'une condition :

```
SI une condition
ALORS on fait des trucs
```

```
SI une condition
ALORS on fait des trucs
SINON on fait autre chose
```

```
SI une condition
ALORS on fait des trucs
SINON SI une autre condition
ALORS on fait des trucs
SINON SI encore une autre condition
ALORS onfait des trucs
SINON on fait des trucs
```

Au niveau de la syntaxe, le python ne précise pas le « ALORS » :

[22]:
```
# Exemple complet pour la syntaxe

age = 253

if age < 6:
    print("Regarde tchoupi")
elif age < 9:
    print("Regarde power rangers")
elif age < 12:
    print("Regarde ta console")
elif age < 16:
    print("Arrête d'être sur ton téléphone et travaille")
elif age < 80:
    print("Abonne toi à Netflix")
else:
    print("Regarde l'inspecteur Derrick")
```

```
Regarde l'inspecteur Derrick
```

Fin de cours étape 3 : faire application correspondante (loi d'Ohm - étape 1)

2.2 Etape 1 : Préparation d'une solution par dissolution

Notions abordées
- Commentaires
- Création d'une variable
- Affectation d'une valeur à une variable
- Affichage de la valeur d'une variable
- Format d'affichage d'un nombre
- Ecriture d'une chaîne de caractères

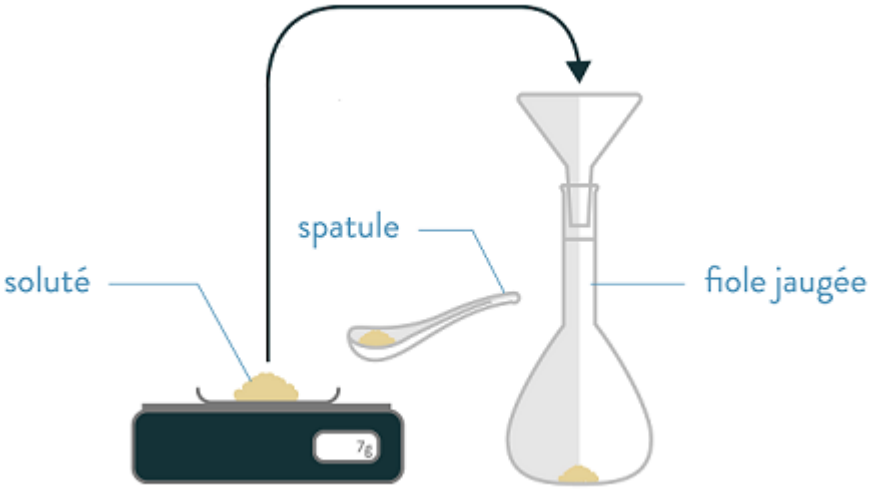
Référence pyspc
- Structure d'un programme
- Affectation variables et affichage des résultats

Consigne :

Etudier le programme ci-dessous puis effectuer la mise en situation présentée dans la dernière cellule.

`Télécharger le pdf` | `Télécharger le notebook` | `Lancer le notebook sur binder (lent)`

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

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
```

(continues on next page)

(suite de la page précédente)

```
# 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

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
```

Mise en situation

Réaliser sous Notebook Jupyter un programme permettant de calculer et d'afficher la valeur de la vitesse d'une voiture (avec 3 chiffres significatifs en écriture scientifique) à partir d'une distance parcourue et d'une durée.

2.3 Etape 2 : Calcul de la structure d'un atome (épisode 1)

Notions abordées
- Types de variables (int, float, string, list)
- Listes

Références pyspc
- Affectation des variables et affichage des résultats
- Les listes

Consigne :

Etudier le programme ci-dessous puis effectuer la mise en situation présentée dans la dernière cellule.

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

Saisir ci-dessous les valeurs du numéro atomique et du nombre de masse de l'élément dont vous voulez calculer la structure, puis exécutez les cellules jusqu'à la fin

[1]:
```
# Entrées : numéro atomique et nombre de masse
# Première possibilité - conseillée : saisie du nombre directement dans la cellule
Z = 6
A = 14
```

[ ]:
```
# Deuxieme possibilité - déconseillée : saisie du nombre dans une ligne de dialogue
# Z = int(input("Numéro atomique : "))
# A = int(input("Nombre de masse : "))
```

[2]:
```
# programme calculant la structure de l'atome


# liste non modifiable (ou tuple) des éléments chimiques
# de la classification périodique

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"]



masse_electron = 9.109e-31
masse_proton = 1.673e-27
masse_neutron = 1.675e-27
```

(continues on next page)

(suite de la page précédente)

```
# calcul des masses
masse_noyau = Z * masse_proton + (A-Z)*masse_neutron
masse = masse_noyau + Z * masse_electron


# 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')
```

```
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:   2.344e-26  kg
la masse de l'atome est:            2.344e-26  kg
```

Mise en situation

Modifier le programme précédent (calcul de la vitesse d'une voiture) afin que celui-ci demande à l'utilisateur les valeurs de la distance et de la durée sous forme de nombres décimaux. Vous pouvez tester les deux manières - mais le input est déconseillé :)

```
[ ]:
```

2.4 Etape 3 : Calcul de la structure d'un atome (épisode 2)

Notions abordées

- Tests
- Initialisation et concaténation de chaînes de caractères
- Conversion de type
- types de variables (int, float, string, list)
- Listes

Références pyspc

- Les tests
- Quelques opérations basiques en python

Consigne :

Etudier le programme ci-dessous puis effectuer la mise en situation présentée dans la dernière cellule.

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

[1]:
```
# programme calculant la structure de l'atome

# Liste non modifiable (ou tuple) des éléments
# chimiques de la classification périodique

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",
```

(continues on next page)

(suite de la page précédente)

```
"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",
```

(continues on next page)

(suite de la page précédente)

```
"Ununoctium Uuo"]

# Entrées
Z = 17
A = 35

# Détermination de la configuration électronique
# (concaténation de chaines de caractères)

if Z <= 2:
    configuration = "1s" + str(Z)
elif Z <=4:
    configuration = "1s" + str(2) + " 2s" + str(Z-2)
elif Z <= 10:
    configuration = ("1s" + str(2) + " 2s" + str(2) +
                     " 2p" + str(Z-4))
elif Z <= 12:
    configuration = ("1s" + str(2) + " 2s" + str(2) +
                     " 2p" + str(6) + " 3s" + str(Z-10))
elif Z <= 18:
    configuration = ("1s" + str(2) + " 2s" + str(2) +
                     " 2p" + str(6) + " 3s" + str(2) +
                     " 3p" + str(Z-12))
else :
    configuration = ("Impossible car limité à des numéros "
                     "atomiques inférieurs ou égal à 18")



# 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)
print("")
# Impression de la configuration électronique
print('{:65}'.format("Le remplissage des sous-couches électroniques "
                     "donne la configuration électronique : ")
      , configuration)
```

```
RESULTATS
Il s'agit de l'élément              Chlore Cl
Le nombre de protons est:           17
Le nombre de neutrons est:          18
Le nombre d'électrons est:          17

Le remplissage des sous-couches électroniques donne la configuration électronique␣
↪:  1s2 2s2 2p6 3s2 3p5
```

Mise en situation

Modifier le programme précédent (calcul de la vitesse d'une voiture) afin d'indiquer à l'utilisateur si le conducteur est en infraction en ville.

2.5 Etape 4 : Loi d'Ohm (épisode 1)

Notions abordées
- Import d'une bibliothèque
- Création d'une liste
- Affichage d'un graphique à l'aide des fonctions de la bibliothèque matplolib.pyplot

Référence pyspc
- Structure d'un programme
- Les listes
- Les graphiques - première partie
- Les fonctions

Consigne :

Etudier le programme ci-dessous puis effectuer la mise en situation présentée dans la dernière cellule.

`Télécharger le pdf` | `Télécharger le notebook` | `Lancer le notebook sur binder (lent)`

On 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.

Exécuter les cellules suivantes.

[1]:
```
# Import de la bibliothèque matplotlib.pyplot
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# Création d'une liste regroupant les valeurs de
# l'intensité I en ampère

I=[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]:
```
# Création d'une liste regroupant les valeurs de
# la tension U en volt.

U=[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. On 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 l'utilité de chaque méthode (fonction) de la bibliothèque matplotlib.pyplot. Noter si besoin des commentaires dans les cellules laissées vides à cet effet.

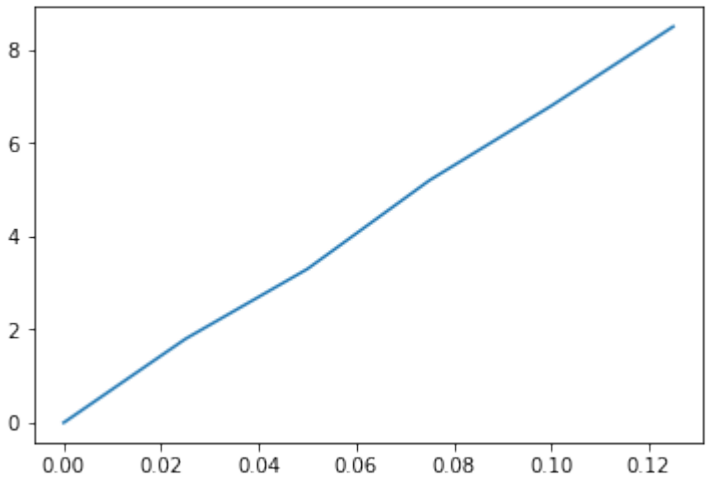
[4]:
```
fig = plt.figure(figsize=(12,10))
```

```
<Figure size 864x720 with 0 Axes>
```

commentaire :

[5]:
```
plt.plot(I,U)
```

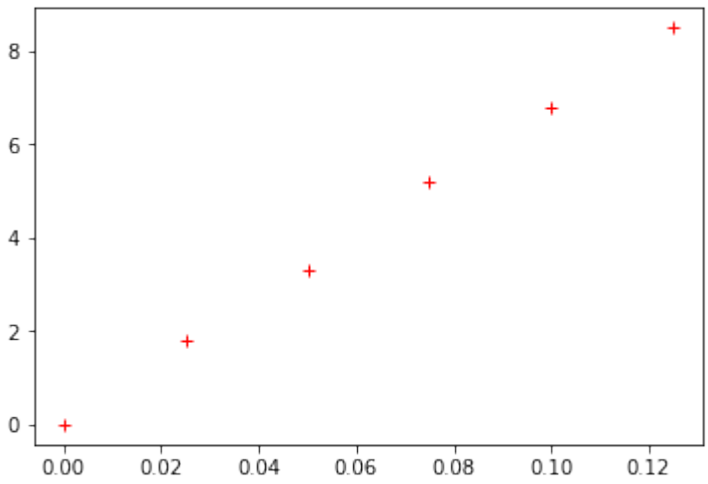
[5]:
```
[<matplotlib.lines.Line2D at 0x7f75add1da58>]
```



commentaire :

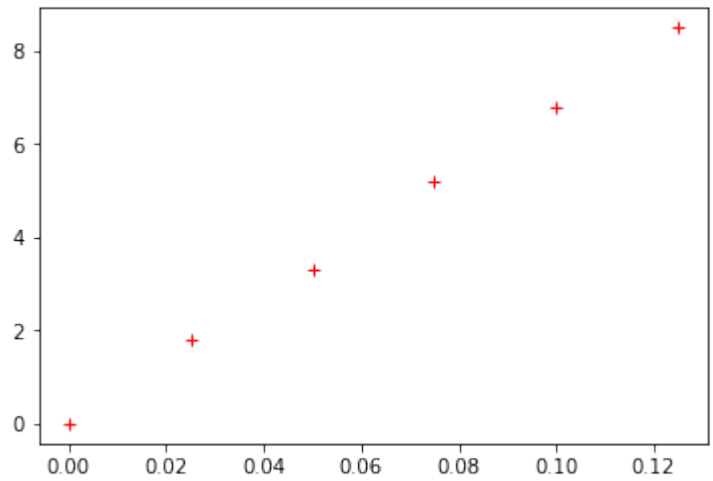
[6]:
```
plt.plot(I,U,'r+')
```

[6]:
```
[<matplotlib.lines.Line2D at 0x7f75adca6f28>]
```



commentaire :

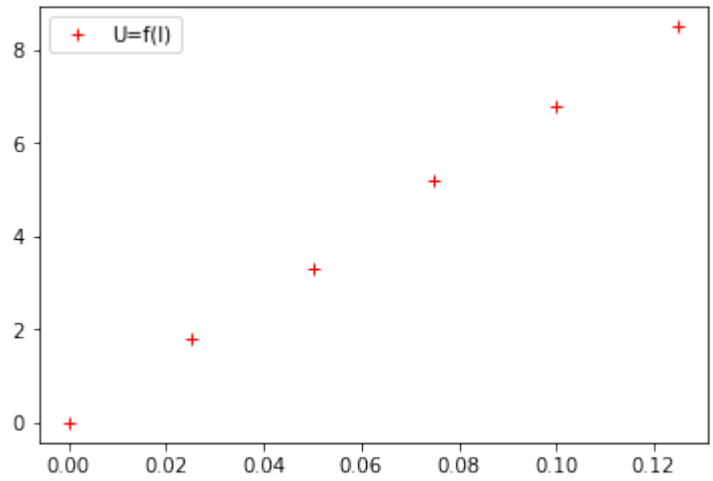
[7]:
```
plt.plot(I,U,'r+')
plt.show()
```



commentaire :

[8]:
```
plt.plot(I,U,'r+',label='U=f(I)')
plt.legend()
```

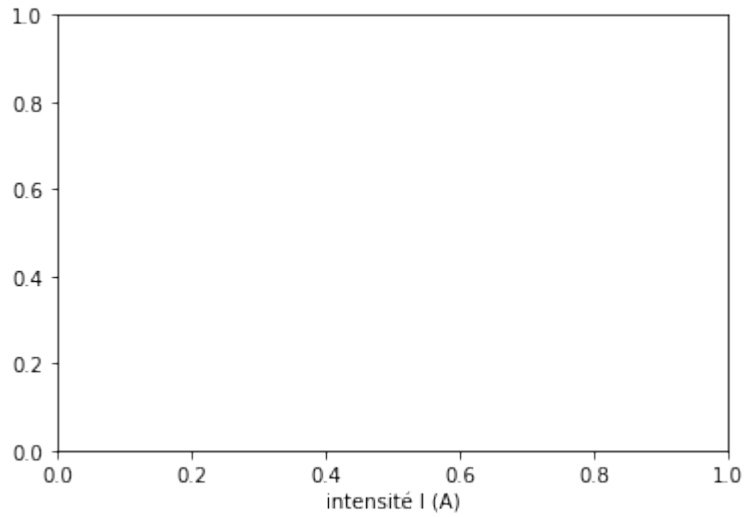
[8]:
```
<matplotlib.legend.Legend at 0x7f75adc5b470>
```



commentaire :

[9]:
```
plt.xlabel("intensité I (A)")
```

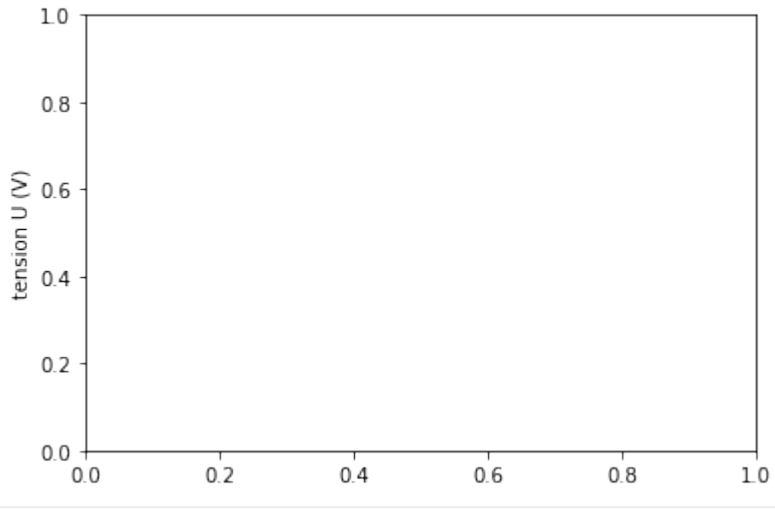
[9]:
```
Text(0.5, 0, 'intensité I (A)')
```



commentaire :

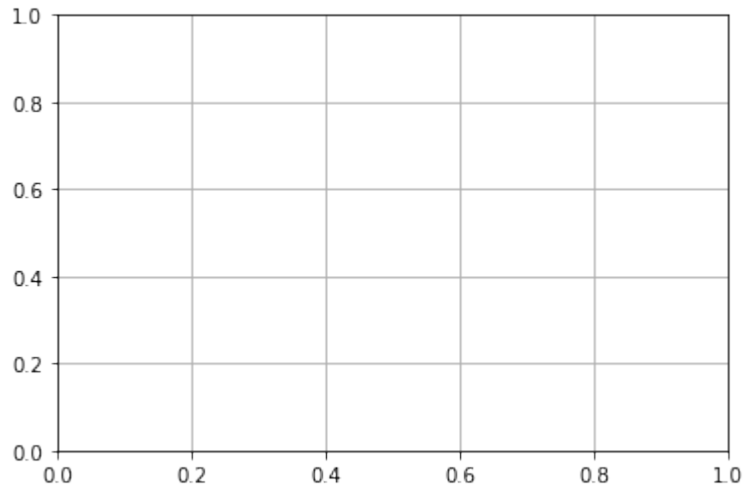
[10]:
```
plt.ylabel("tension U (V)")
```

[10]:
```
Text(0, 0.5, 'tension U (V)')
```



commentaire :

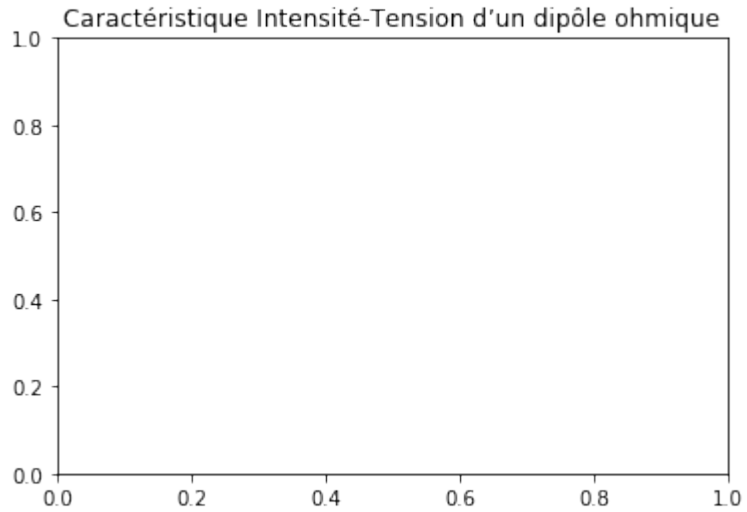
[11]:
```
plt.grid()
```



commentaire :

[12]:
```
plt.title("Caractéristique Intensité-Tension "
          "d'un dipôle ohmique")
```

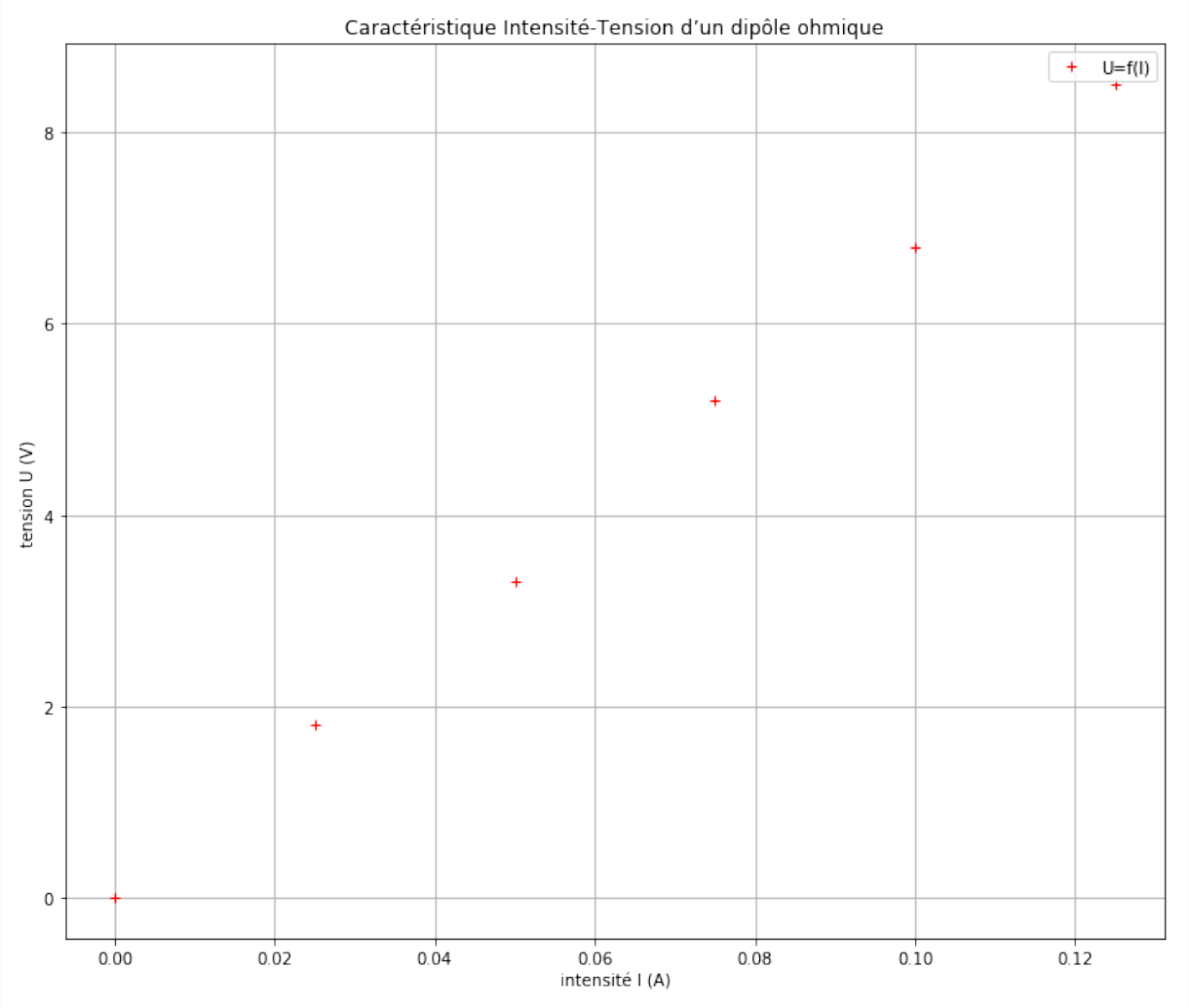
[12]:
```
Text(0.5, 1.0, 'Caractéristique Intensité-Tension d'un dipôle ohmique')
```



commentaire :

4. Exécutez maintenant le programme en entier !

[13]:
```
import 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]
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()
```



Mise en situation

- Commenter le programme « Etape 4 : La loi d'Ohm (épisode 1) »
- Faire une copie de ce notebook Notebook « Etape 4 : La loi d'Ohm (épisode 1) » puis modifier le programme afin d'afficher la courbe I=f(U) avec des cercles bleus

2.6 Etape 5 : Loi d'Ohm (épisode 2)

Notions abordées

- Création d'un tableau à l'aide de la bibliothèque numpy
- Appel et utilisation de la fonction polyfit de la bibliothèque numpy
- Création d'un tableau numpy à partir d'un calcul faisant intervenir d'autres tableaux numpy
- Affichage d'un tableau numpy
- Appel et affichage des valeurs d'un tableau numpy
- Affichage de deux courbes sur un même graphique
- Pointage et zoom sur une courbe

Référence pyspc

- Les tableaux numpy
- Les modélisations
- Les fonctions
- Les graphiques - première partie
- Les graphiques - zoom et pointeur

Consigne :

Etudier le programme ci-dessous puis effectuer la mise en situation présentée dans la dernière cellule.

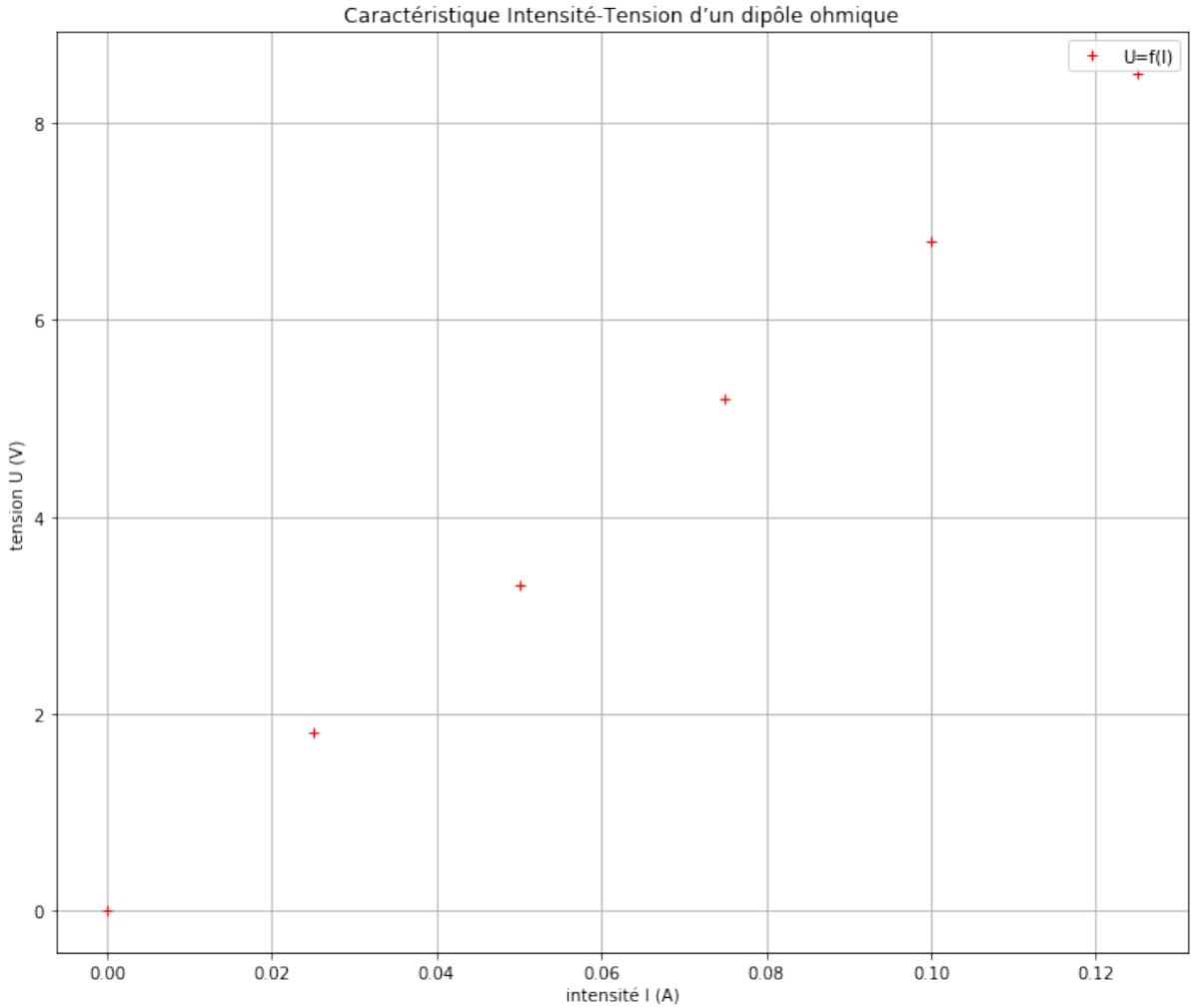
`Télécharger le pdf` | `Télécharger le notebook` | `Lancer le notebook sur binder (lent)`

On souhaite modéliser 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.

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

[2]:
```
# Création des tableaux de valeurs avec la bibliothèque numpy

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])
```

[3]:
```
# Reprise du programme vu précédemment et permettant d'afficher
# la caractéristique d'un dipole ohmique.

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()
```



Il s'agit maintenant de modéliser la courbe obtenue.

Exécutez le programme ci-dessous permettant de modéliser la courbe obtenue par une droite.

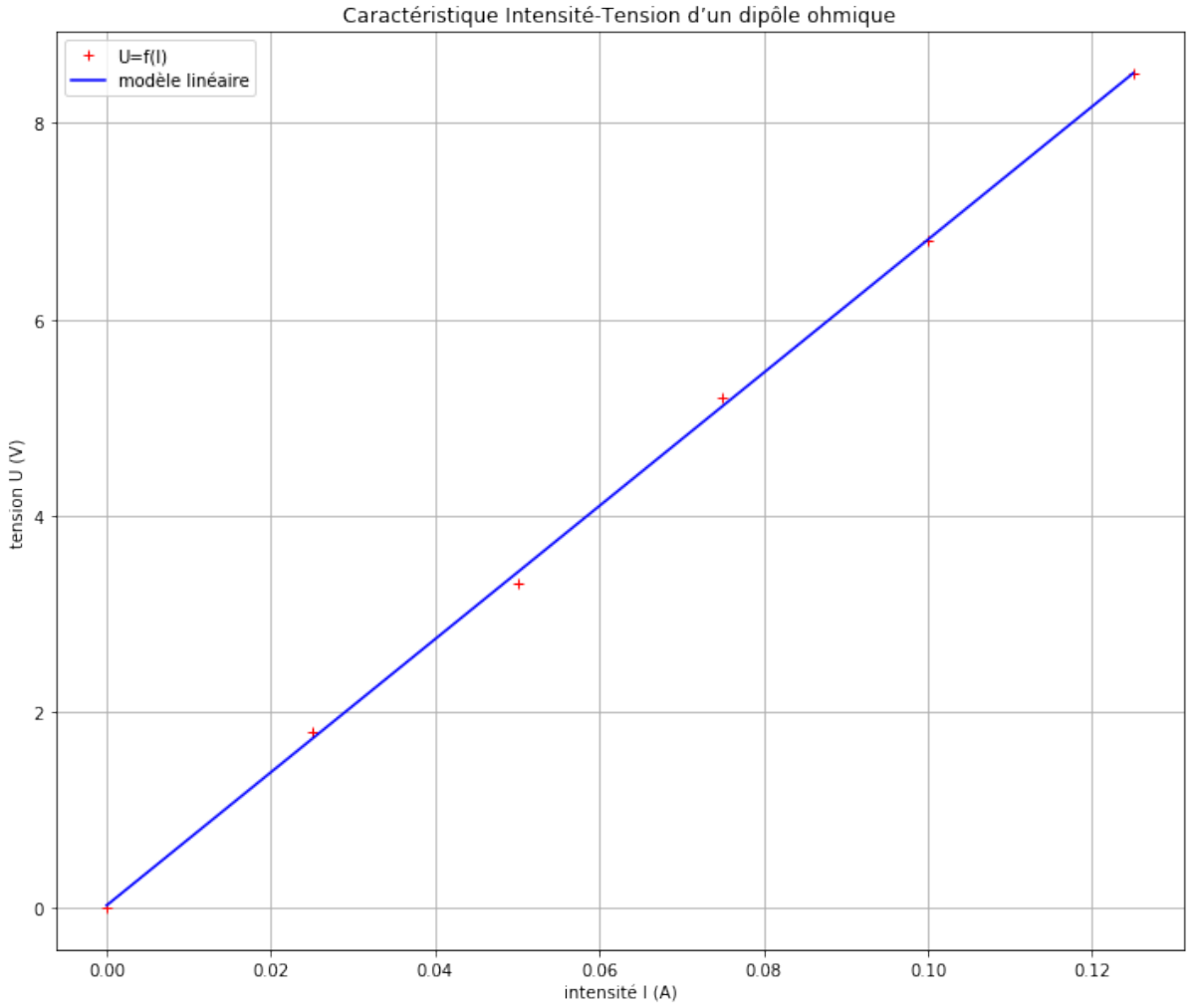
[4]:
```
coeff=np.polyfit(I, U,1)
# La fonction polyfit (inclue dans la bibliothèque numpy)
# retourne un tableau numpy a une dimension (nommée ici coeff)
# contenant les coefficients d'un polynome U=f(I) de degré 1 (ici).

# Affichage du tableau coeff
print(coeff)

# Affichage des valeurs contenues dans le tableau avec une décimale
print ('{0:.1f}'.format(coeff[0]),
       '{0:.1f}'.format(coeff[1]))

# Création puis affichage d'un tableau numpy Umodel regroupant
# les valeurs de la tension modélisée par une fonction affine.
# Cette opération dite vectorisée ne peut pas se faire sur des listes,
# d'où la nécessité de créer des tableaux numpy.
Umodel = coeff[0]*I+coeff[1]

print(Umodel)

# Affichage de l'équation de la droite modélisée
print('U={0:.1f}'.format(coeff[0]),'x I')
```

```
[6.78857143e+01 2.38095238e-02]
67.9 0.0
[0.02380952 1.72095238 3.41809524 5.1152381  6.81238095 8.50952381]
U=67.9 x I
```

1. Que représentent les trois paramètres ordonnés de la fonction polyfit ?
2. Que représente coeff[0] ?
3. Que représente coeff[1] ?

Affichez la droite modélisée grâce au programme ci-dessous.

[5]:
```
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()
```



Si vous désirez faire un zoom ou utiliser un pointeur sur la courbe cela est possible (uniquement dans le logiciel Jupyter Notebook (ENT, Anaconda) mais pas dans le logiciel JupyterLab) en important la bibliothèque ipywidgets qui permet d'afficher une fenêtre interactive.

Attention,ne pas écrire la ligne de code %matplotlib inline dans le programme pour voir s'afficher la fenêtre interactive. Ici, il sera nécessaire d'exécuter la cellule deux fois de suite afin de voir s'afficher cette fenêtre et la barre d'outil qui permet d'interagir avec la courbe.

[7]:
```
import ipywidgets as widgets
%matplotlib widget

fig = plt.figure(figsize=(6,5))
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()
```

```
Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home', 'home'),␣
↪('Back', 'Back to previous ...
```

Mise en situation

- Ouvrir le programme « Activité : Modélisation d'un mouvement parabolique » puis suivre les instructions pour compléter le programme.
- Vérifier votre travail si besoin avec le notebook « Correction : Modélisation d'un mouvement parabolique »

Liens vers les documents

2.6.1 Etape 5 : Activité : Modélisation d'un mouvement parabolique

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

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

[ ]:
```
# Création des tableaux de valeurs avec la bibliothèque numpy

t=np.array([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])

x=np.array([-0.003, 0.065, 0.140, 0.214, 0.287, 0.362, 0.435,
            0.514, 0.584, 0.663, 0.739, 0.815, 0.890, 0.9662,
            1.039, 1.115, 1.191, 1.270, 1.340])

y=np.array([0.0, 0.143, 0.267, 0.376, 0.472, 0.553, 0.618, 0.666,
            0.694, 0.713, 0.713, 0.696, 0.660, 0.618, 0.553, 0.469,
            0.374, 0.261, 0.135])
```

Ecrire ci-dessous les lignes de code permettant d'afficher sur un même graphique les courbes x=f(t) et y=f(t)

[ ]:
```

```

Ecrire ci-dessous les lignes de code permettant de modéliser correctement la courbe y=f(t) et de créer un tableau contenant le modèle.

[ ]:
```

```

Ecrire ci-dessous les lignes de code permettant d'afficher la courbe y=f(t) ainsi que la courbe du modèle.

[ ]:

Télécharger le pdf de correction | Télécharger le notebook de correction | Lancer le notebook de correction sur binder (lent)

2.6.2 Etape 5 : Activité : Correction : Modélisation d'un mouvement parabolique

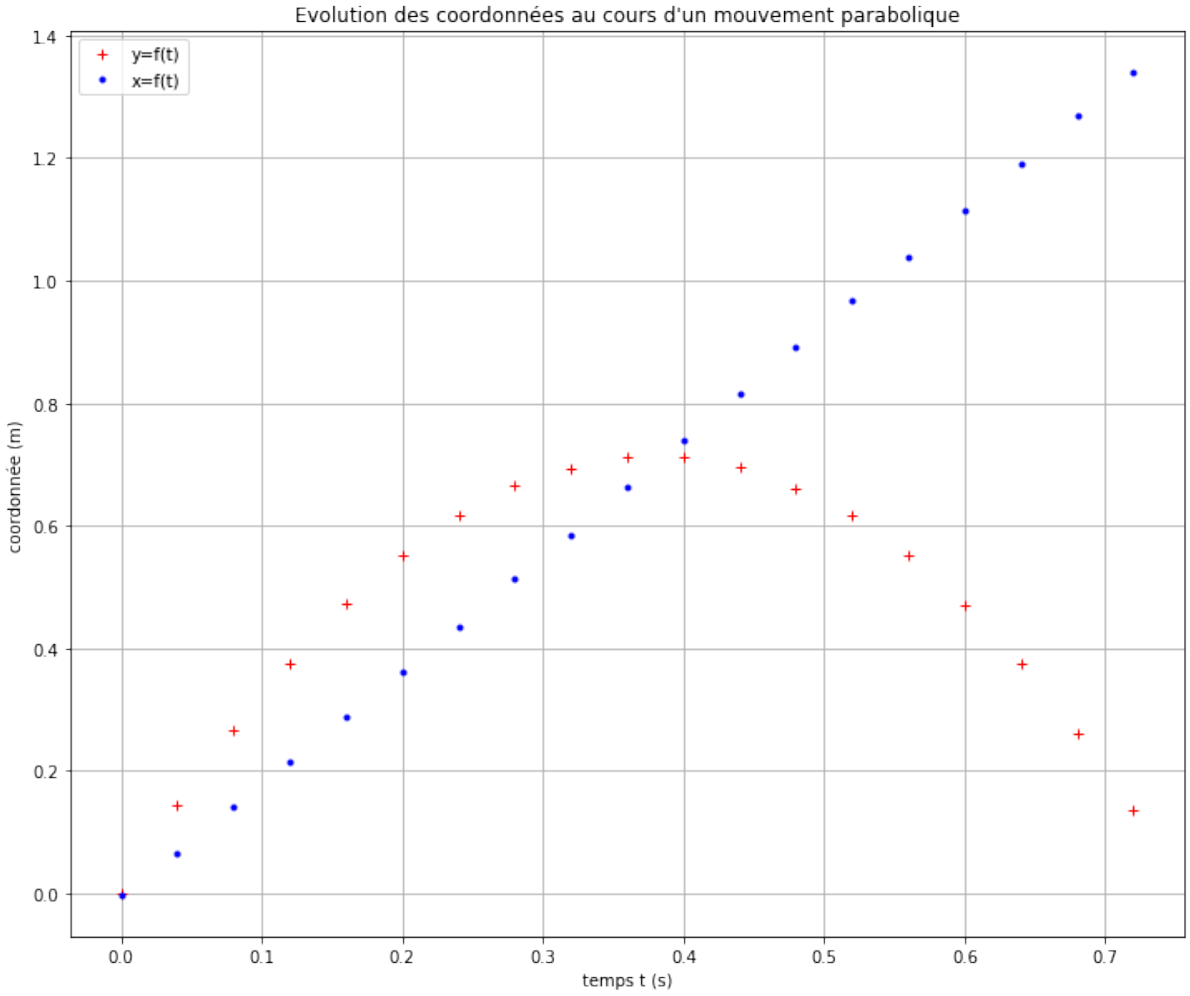
Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

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

[2]:
```
# Création des tableaux de valeurs avec la bibliothèque numpy

t=np.array([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])

x=np.array([-0.003, 0.065, 0.140, 0.214, 0.287, 0.362, 0.435,
            0.514, 0.584, 0.663, 0.739, 0.815, 0.890, 0.9662,
            1.039, 1.115, 1.191, 1.270, 1.340])

y=np.array([0.0, 0.143, 0.267, 0.376, 0.472, 0.553, 0.618, 0.666,
            0.694, 0.713, 0.713, 0.696, 0.660, 0.618, 0.553, 0.469,
            0.374, 0.261, 0.135])
```

Ecrire ci-dessous les lignes de code permettant d'afficher sur un même graphique les courbes x=f(t) et y=f(t)

[3]:
```
# Reprise du programme vu précédemment
fig = plt.figure(figsize=(12,10))
plt.plot(t,y,'r+',label='y=f(t)')
plt.plot(t,x,'.b',label='x=f(t)')
plt.legend()
plt.xlabel("temps t (s)")
plt.ylabel("coordonnée (m)")
plt.grid()
plt.title("Evolution des coordonnées au cours d'un mouvement parabolique")
plt.show()
```



Ecrire ci-dessous les lignes de code permettant de modéliser correctement la courbe y=f(t) et de créer une liste contenant le modèle.

[4]:
```
coeff=np.polyfit(t, y,2)

# Affichage du tableau coeff
print(coeff)

# Affichage des valeurs contenues dans le tableau avec une décimale
print ('{0:.1f}'.format(coeff[0]),
       '{0:.1f}'.format(coeff[1]),
       '{0:.1f}'.format(coeff[2]))

# Création puis affichage d'un tableau Ymodel regroupant les valeurs de la
# coordonnée y modélisée par une fonction polynôme de degré 2.
Ymodel = coeff[0]*t**2+coeff[1]*t+coeff[2]
```

```
[-4.97991302e+00  3.77571281e+00 -2.13984962e-03]
-5.0 3.8 -0.0
```

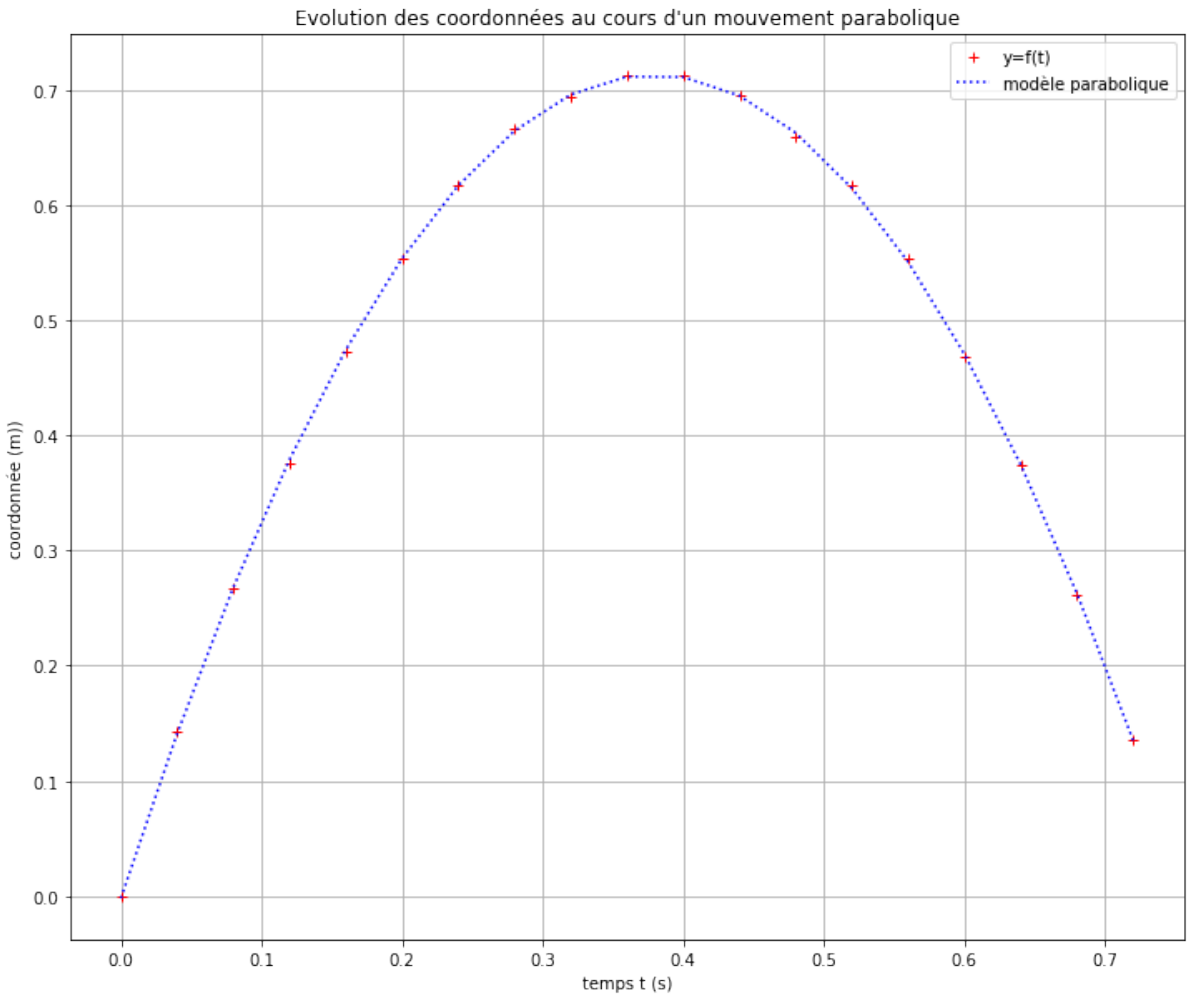
Ecrire ci-dessous les lignes de code permettant d'afficher la courbe y=f(t) ainsi que la courbe du modèle.

[8]:
```
fig = plt.figure(figsize=(12,10))
plt.plot(t,y,'r+',label='y=f(t)')
plt.plot(t,Ymodel,'b:',label='modèle parabolique')
plt.legend()
```

(continues on next page)

(suite de la page précédente)

```
plt.xlabel("temps t (s)")
plt.ylabel("coordonnée (m))")
plt.grid()
plt.title("Evolution des coordonnées au cours d'un mouvement parabolique")
plt.show()
```



Télécharger le notebook Lancer le notebook sur binder (lent)

Télécharger le notebook de correction Lancer le notebook sur binder (lent)

2.7 Etape 6 : Calcul des coordonnées d'un vecteur vitesse

Notions abordées

- Boucle for
- Ajout d'une valeur dans une liste
- Parcours des valeurs d'une liste
- Longueur d'une liste
- Création et intérêt d'une fonction

Références pyspc

- Les boucles
- Les listes
- Les fonctions

Consigne :

Etudier les programmes ci-dessous puis effectuer la mise en situation présentée dans la dernière cellule.

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

Etude préliminaire

Exécuter la cellule ci-dessous puis analyser ce qu'elle renvoie afin de comprendre les différentes instructions posées.

[1]:
```
Liste=["a",11,12,13,14,15,"b",17,18,"c"]

print(Liste)
print('')
print("nombre d'éléments dans la liste=",len(Liste))
print('\n')
print(Liste[0])
print(Liste[3],'\n')
for i in range(10):
    print(i,'',Liste[i])
print('\n')
for i in range (2,9):
    print(i,'',Liste[i])
Liste.append("d")
print("")
print(Liste)
```

```
['a', 11, 12, 13, 14, 15, 'b', 17, 18, 'c']

nombre d'éléments dans la liste= 10


a
13

0  a
1  11
2  12
3  13
4  14
5  15
6  b
7  17
8  18
9  c


2  12
3  13
4  14
5  15
6  b
7  17
8  18

['a', 11, 12, 13, 14, 15, 'b', 17, 18, 'c', 'd']
```

Exécuter maintenant les deux cellules ci-dessous afin de comprendre comment créer puis appeler une fonction.

[2]:
```
# Création de la fonction f dont les paramètres d'entrée ordonnés
# sont x et y et qui retourne le paramètre de sortie z

def f(x,y):
```

(continues on next page)

(suite de la page précédente)

```
    z=2*x+y
    return z
```

[3]:
```
# Différents appels possibles de la fonction

u=f(2,3)
print(u)

toto=3
titi=4
v=f(toto,titi)
print (v)

# ci-dessous, la valeur retournée par la fonction est affichée
# mais n'est pas affectée à une variable et ne peut
# donc pas être réutilisée plus loin dans le programme

print (f(titi,toto))
```

```
7
10
11
```

Détermination des coordonnées d'un vecteur vitesse

[4]:
```
# Création des listes contenant les valeurs du temps
# et des coordonnées du vecteur position

t=[0.0, 0.04, 0.08, 0.12, 0.16, 0.2, 0.24, 0.28, 0.32,
   0.36, 0.4, 0.44, 0.48, 0.52, 0.56, 0.6, 0.64, 0.68, 0.72]

x=[-0.003, 0.065, 0.140, 0.214, 0.287, 0.362, 0.435,
   0.514, 0.584, 0.663, 0.739, 0.815, 0.890, 0.9662,
   1.039, 1.115, 1.191, 1.270, 1.340]

y=[0.0, 0.143, 0.267, 0.376, 0.472, 0.553, 0.618,
   0.666, 0.694, 0.713, 0.713, 0.696, 0.660, 0.618,
   0.553, 0.469, 0.374, 0.261, 0.135]
```

[5]:
```
# Création d'une liste contenant les valeurs de
# la coordonnée vx du vecteur vitesse

vx=[]
for i in range (len(x)-1):
    vxi=(x[i+1]-x[i])/(t[i+1]-t[i])
    vx.append(vxi)
print(vx)
```

```
[1.7000000000000002, 1.8750000000000002, 1.8499999999999999, 1.8249999999999993, 1.
↪875, 1.825000000000001, 1.9749999999999985, 1.7499999999999996, 1.
↪9750000000000028, 1.8999999999999972, 1.9, 1.8750000000000027, 1.
↪9049999999999967, 1.8199999999999978, 1.9000000000000052, 1.9, 1.
↪9749999999999972, 1.7500000000000049]
```

[6]:
```
# Création d'une liste contenant les valeurs
# de la coordonnée vy du vecteur vitesse

vy=[]
for i in range (len(y)-1):
    vyi=(y[i+1]-y[i])/(t[i+1]-t[i])
    vy.append(vyi)
```

(continues on next page)

(suite de la page précédente)

```
print(vy)
```

```
[3.5749999999999997, 3.1000000000000005, 2.725, 2.399999999999999, 2.
↪0250000000000012, 1.6249999999999996, 1.2, 0.6999999999999982, 0.
↪47500000000000064, 0.0, -0.4250000000000006, -0.8999999999999985, -1.05, -1.
↪6249999999999973, -2.100000000000006, -2.3749999999999973, -2.824999999999997,␣
↪-3.150000000000006]
```

Mise en situation

Faire une copie de ce notebook puis modifier le programme en créant une fonction permettant d'éviter la répétition des lignes de code lors du calcul des coordonnées du vecteur vitesse

2.8 Etape 7 : Tracé d'un vecteur vitesse

Notions abordées
— Tracés de vecteurs
Référence pyspc
— Les graphiques (deuxième partie)
Consigne :

Etudier le programme ci-dessous puis effectuer la mise en situation présentée dans la dernière cellule.

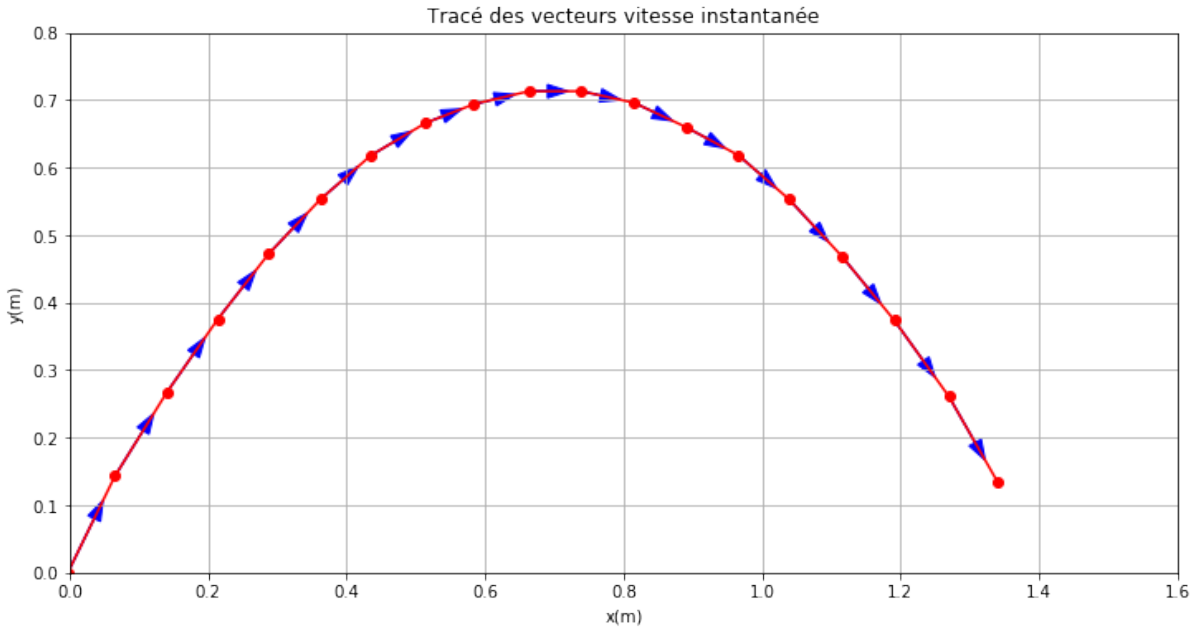
Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

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

# Création des listes contenant les valeurs du temps
# et des coordonnées du vecteur position

t=[0.0, 0.04, 0.08, 0.12, 0.16, 0.2, 0.24, 0.28, 0.32,
   0.36, 0.4, 0.44, 0.48, 0.52, 0.56, 0.6, 0.64, 0.68, 0.72]

x=[-0.003, 0.065, 0.140, 0.214, 0.287, 0.362, 0.435,
   0.514, 0.584, 0.663, 0.739, 0.815, 0.890, 0.9662,
   1.039, 1.115, 1.191, 1.270, 1.340]

y=[0.0, 0.143, 0.267, 0.376, 0.472, 0.553, 0.618, 0.666,
   0.694, 0.713, 0.713, 0.696, 0.660, 0.618, 0.553,
   0.469, 0.374, 0.261, 0.135]
```

[2]:
```
# Création d'une fonction permettant de calculer
# les valeurs d'une coordonnée du vecteur vitesse

def vitesse(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
```

[3]:
```
vx=vitesse(t,x)
vy=vitesse(t,y)
```

[4]:
```
fig = plt.figure(1,figsize=(12,6))
plt.plot(x,y,'ro-')
plt.xlim(0,1.6)
plt.ylim(0,0.8)
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 instantanée")
plt.show()
```



Mise en situation

Télécharger, faire une copie de ce notebook « Tracé d'un vecteur vitesse » puis modifier les paramètres de la fonction permettant de tracer les vecteurs afin de comprendre leur utilité.

2.9 Etape 8 : Importer les données numériques d'un tableur scientifique dans un programme python

Notions abordées

— Import de données numériques à partir d'un tableur

Référence pyspc

— Import de données numériques

Consigne :

Télécharger, faire une copie du notebook « Importer les données numériques d'un tableur scientifique dans un programme python » puis répondre aux questions posées dans le notebook. Enfin effectuer la mise en situation présentée dans la dernière cellule.

`Télécharger le pdf | Télécharger le notebook | Télécharger le 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.

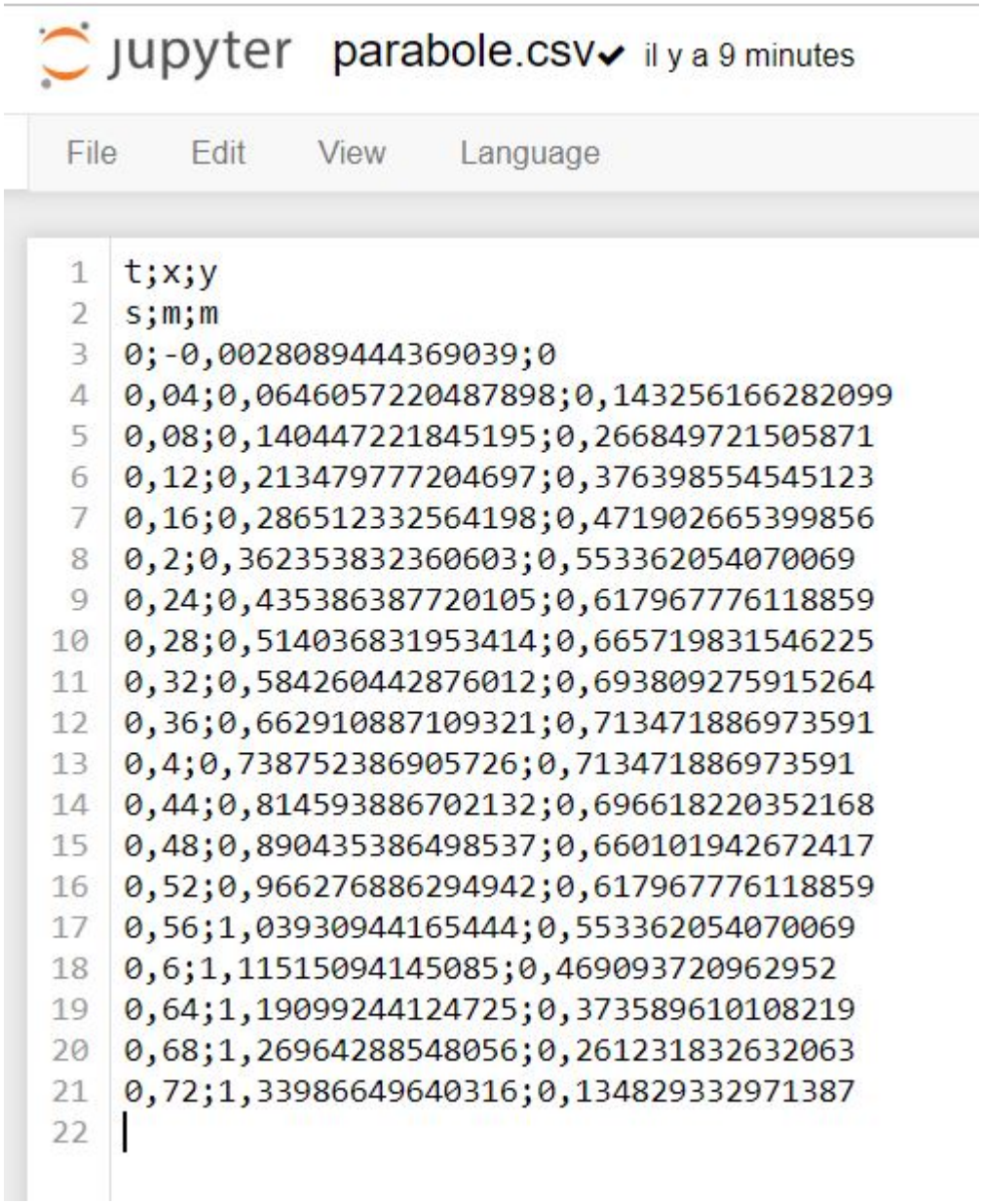
1. Enregistrer ou exporter le fichier contenant votre tableau de données sous format .csv (ou .txt pour Avistep) dans le dossier contenant votre notebook (fichier .ipynb) ou votre programme python (fichier.py). Attention, pour l'utilisation avec l'ENT Nero version 2018, petite subtilité à la fin.
   — 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 csv)

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

import csv
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

[2]:
```
# Ceci est une fonction (nommée charge_fichier_csv)
# qui sera appelée si besoin est, par le programme principal
# situé dans une autre cellule, plus loin

# Cette fonction aura pour arguments un fichier qui lui est
# fourni par le programme principal (paramètre écrit entre
# parenthèses et nommé ici cheminfichier), le délimiteur
# de cellules et le nombre de lignes d'en-tête N. Il renvoie un
```

(continues on next page)

(suite de la page précédente)

```
# résultat au programme principal grâce à l'instruction return
# (ici un tableau de valeurs nommé tab)

def charge_fichier_csv(cheminfichier, delimiter=";",N=0):
    with open(cheminfichier, 'r', encoding='utf-8') as f :
        rfichier = csv.reader(f, delimiter=delimiter)
        tab=[]
        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(tab) <= i:
                        X = []
                        tab.append(X)
                    try:
                        tab[i].append(float(row[i].replace(",",'.')))
                    except ValueError:
                        print('erreur:contenu de cellule non numérique')
                        continue

    return tab
```

[3]:
```
# Programme principal
# ouverture, lecture et affichage du fichier "parabole. csv" .
# 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


table = charge_fichier_csv("parabole.csv",delimiter=";",N=2)
t=table[0]
print(t)
x=table[1]
print(x)
y=table[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]
```

Répondre aux questions suivantes :

- Quelle ligne de code dans le programme principal permet d'appeler la fonction ?
- Comment s'appelle le fichier envoyé à la fonction par le programme principal ?
- Que doit-on écrire entre parenthèses lors de l'appel de la fonction pour pouvoir accéder au fichier ?
- Comment s'appelle les paramètres correspondant dans la fonction ?
- Comment s'appelle la variable contenant le tableau renvoyé par la fonction dans le programme principal ?

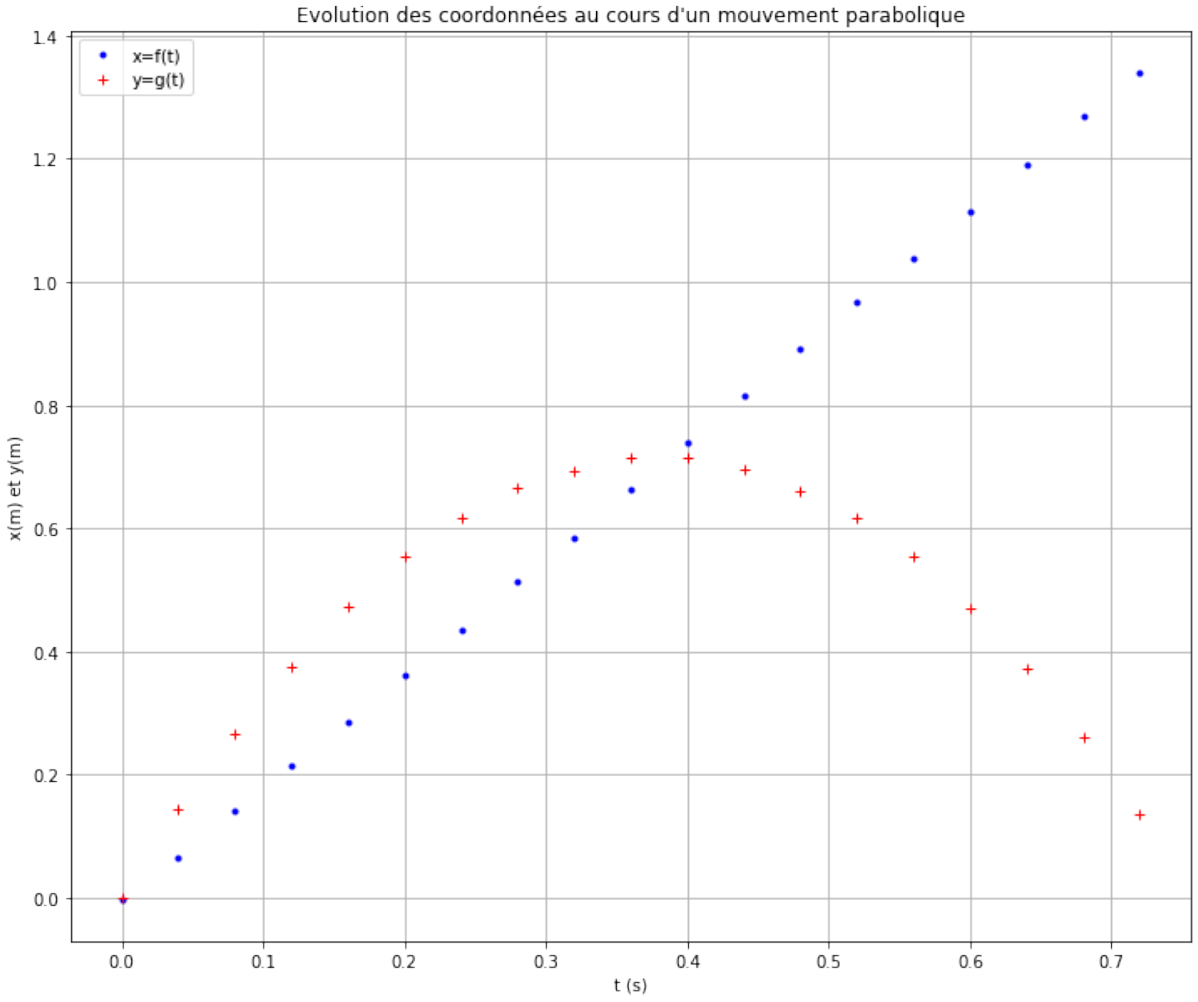
La suite du notebook vous donne un aperçu de ce qu'il est possible de faire avec les données importées.

[4]:
```
# Affichage des courbes x=f(t) et y=g(t)
fig = plt.figure(figsize=(12,10))
plt.plot(t,x,'.b',label='x=f(t)')
plt.plot(t,y,'r+',label='y=g(t)')
```

(continues on next page)

(suite de la page précédente)

```
plt.legend()
plt.xlabel("t (s)")
plt.ylabel("x(m) et y(m)")
plt.grid()
plt.title("Evolution des coordonnées au cours d'un mouvement parabolique")
plt.show()
```



[5]:
```
# Modélisation de la courbe x=f(t)

t=np.array(t)

coeffx=np.polyfit(t, x,1)

# Affichage du tableau coeffx
print('coefficients de Xmodel:', coeffx)

# Affichage des valeurs contenues dans la liste avec une décimale
print ('coeffx[0]=','{0:.1f}'.format(coeffx[0]),'\n',
       'coeffx[1]=','{0:.1f}'.format(coeffx[1]))

# Création puis affichage d'un tableau Xmodel regroupant les valeurs
# de la coordonnée x modélisée par une fonction affine.

Xmodel = coeffx[0]*t+coeffx[1]
print('Le tableau contenant les valeurs de Xmodel est:\n',Xmodel)
```

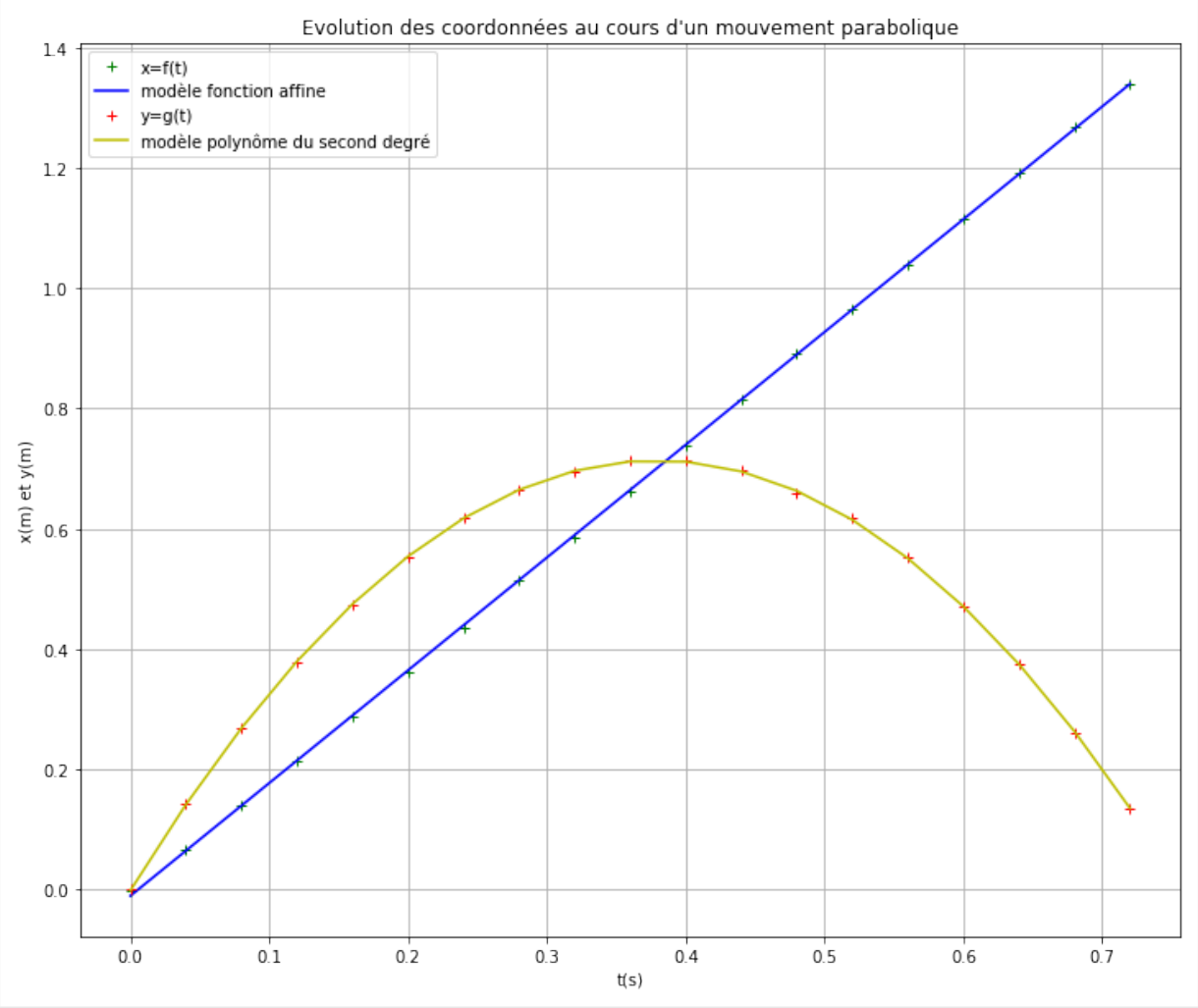
(continues on next page)

(suite de la page précédente)

```
# Modélisation de la courbe y=g(t)

coeffy=np.polyfit(t, y,2)

# Affichage de la liste coeff
print('coefficients de Ymodel:', coeffy)

# Affichage des valeurs contenues dans la liste avec une décimale
print ('coeffy[0]=','{0:.1f}'.format(coeffy[0]),'\n',
       'coeffy[1]=','{0:.1f}'.format(coeffy[1]),'\n',
       'coeffy[2]=','{0:.1f}'.format(coeffy[2]))

# Création puis affichage d'un tableau Ymodel regroupant les valeurs
# de la coordonnée y modélisée par une fonction polynome du second degré.

Ymodel = coeffy[0]*t**2+coeffy[1]*t+coeffy[2]
print('Le tableau contenant les valeurs de Ymodel est:\n',Ymodel)
```

```
coefficients de Xmodel: [ 1.8766952  -0.01107315]
coeffx[0]= 1.9
 coeffx[1]= -0.0
Le tableau contenant les valeurs de Xmodel est:
 [-0.01107315  0.06399465  0.13906246  0.21413027  0.28919808  0.36426589
  0.43933369  0.5144015   0.58946931  0.66453712  0.73960493  0.81467273
  0.88974054  0.96480835  1.03987616  1.11494397  1.19001177  1.26507958
  1.34014739]
coefficients de Ymodel: [-4.98167828e+00  3.77690491e+00 -2.15000401e-03]
coeffy[0]= -5.0
 coeffy[1]= 3.8
 coeffy[2]= -0.0
Le tableau contenant les valeurs de Ymodel est:
 [-0.00215     0.14095551  0.26811965  0.37934242  0.47462382  0.55396385
  0.61736251  0.66481979  0.69633571  0.71191026  0.71154343  0.69523524
  0.66298568  0.61479474  0.55066244  0.47058876  0.37457371  0.2626173
  0.13471951]
```

[6]:
```
# Affichage des courbes modélisées

fig = plt.figure(figsize=(12,10))
plt.plot(t,x,'g+',label='x=f(t)')
plt.plot(t,Xmodel,'b-',label='modèle fonction affine')
plt.plot(t,y,'r+',label='y=g(t)')
plt.plot(t,Ymodel,'y-',label='modèle polynôme du second degré')
plt.legend()
plt.xlabel("t(s)")
plt.ylabel("x(m) et y(m)")
plt.grid()
plt.title("Evolution des coordonnées au cours d'un mouvement parabolique")
plt.show()
```



Mise en situation

- Télécharger le programme complet niveau seconde : étude de la chute d'une balle lachée par un cycliste en mouvement.
- Télécharger le fichier « chute_balle.csv » dans le même dossier.
- Puis étudier ce programme.

Liens vers les documents

2.9.1 Etape 8 : Application : Etude de la chute d'une balle lachée par un cycliste en mouvement

`Télécharger le pdf` | `Télécharger le notebook`, `le fichier csv` et `la video` | `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(cheminfichier, delimiter=";",N=0):
    with open(cheminfichier, 'r', encoding='utf-8') as f :
        rfichier = csv.reader(f, delimiter=delimiter)
        tab=[]
```

(continues on next page)

(suite de la page précédente)

```
        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(tab) <= i:
                        X = []
                        tab.append(X)
                    try:
                        tab[i].append(float(row[i].replace(",",'.')))
                    except ValueError:
                        print('erreur:contenu de cellule non numérique')
                        continue

    return tab
```

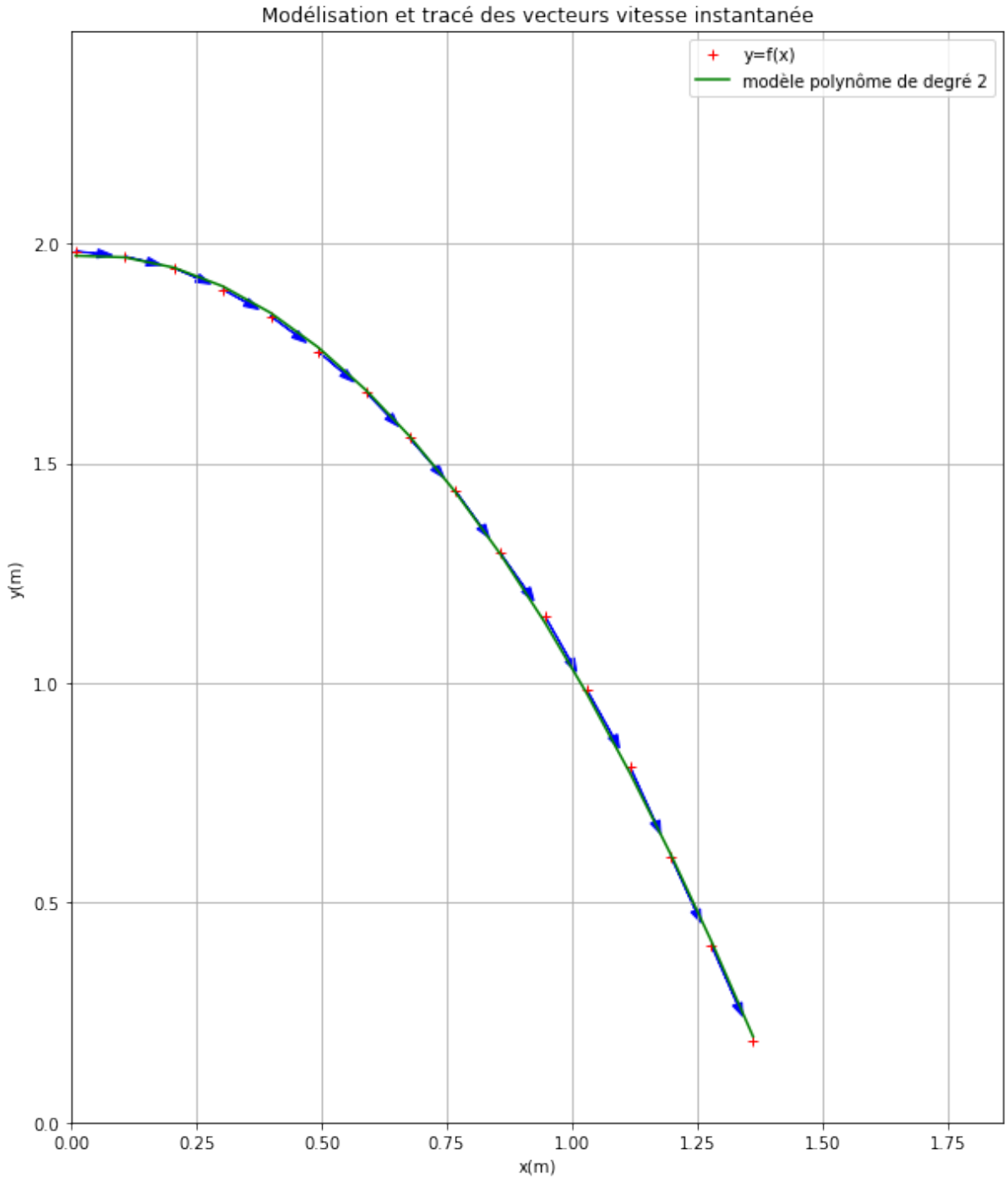
[3]:
```
def model_polynome_deg2(u,v):
    u=np.array(u)
    v=np.array(v)
    coeff=np.polyfit(u,v,2)
    vmodel=coeff[0]*u**2+coeff[1]*u+coeff[2]
    return vmodel
```

[4]:
```
def derivee(t,u):
    du=[]
    for i in range (len(u)-1):
        dui=(u[i+1]-u[i])/(t[i+1]-t[i])
        du.append(dui)
    return du
```

[5]:
```
def graphvect(x,y,ymodel,vx,vy):
    fig = plt.figure(1,figsize=(10,12))
    plt.plot(x,y,'r+',label='y=f(x)')
    plt.plot(x,ymodel,'g-',label='modèle polynôme de degré 2')
    plt.xlim(0,max(x)+0.5)
    plt.ylim(0,max(y)+0.5)
    plt.grid()
    plt.xlabel("x(m)")
    plt.ylabel("y(m)")
    plt.legend()
    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("Modélisation et tracé des vecteurs vitesse instantanée")
    plt.show()
```

[6]:
```
tableau = charge_fichier_csv("chute_balle.csv",delimiter=";",N=1)
t=tableau[0]
print(t)
x=tableau[1]
print(x)
y=tableau[2]
print(y)
ymodel=model_polynome_deg2(x,y)
print(ymodel)
vx=derivee(t,x)
vy=derivee(t,y)
graphvect(x,y,ymodel,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]
[1.97254171 1.96885614 1.94560635 1.90279235 1.84252803 1.7638334
 1.66670847 1.56241984 1.43438997 1.29383467 1.13701536 0.97558948
 0.7936194  0.61003832 0.41318878 0.19532115]
```



[ ]:

Télécharger le notebook

Lancer le notebook sur binder (lent)

Télécharger le csv

2.10 Etape 9 : Etude de l'évolution d'un système chimique (version élève)

Notions abordées
- Boucle while
- Opérateurs logiques

Référence pyspc
- Les boucles
- Quelques opérations basiques en python

Consigne et mise en situation
- Télécharger, faire une copie de ce notebook «Etude de l'évolution d'un système chimique (version élève)», étudier le programme puis répondre aux questions posées.
- Vérifier votre travail si besoin avec le notebook «Etude de l'évolution d'un système chimique (version professeur)»

Correction

2.10.1 Etape 9 : Correction : 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
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

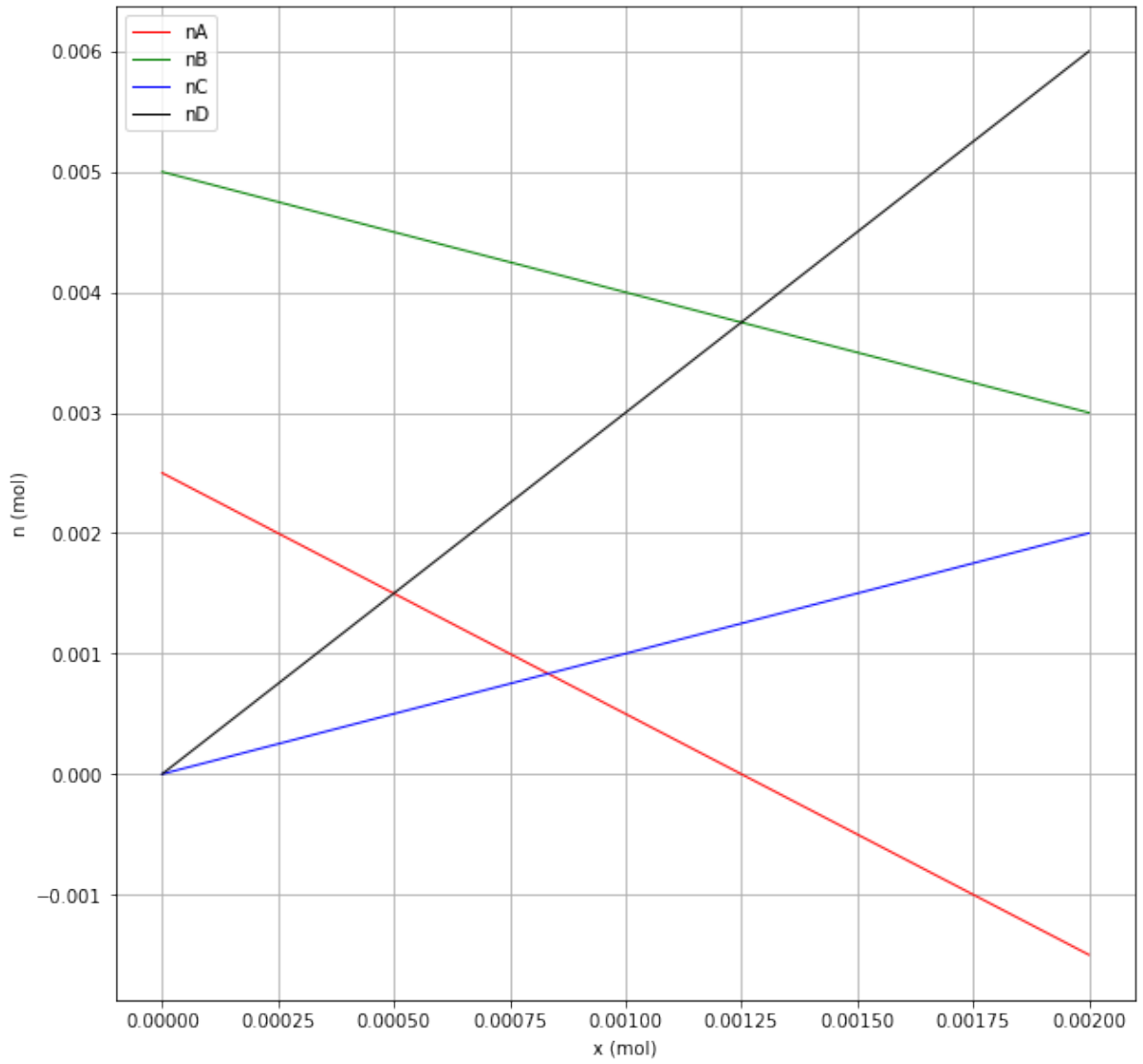
La ligne de code 2 ci-dessous permet d'afficher le graphe de l'évolution de la quantité de matière de A en fonction de l'avancement X.

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"
```

(continues on next page)

(suite de la page précédente)

```
       " 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....

Lien fichiers - version élève

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

Lien fichiers - version prof

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 : ",
```

(continues on next page)

(suite de la page précédente)

```
      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)
```

(continues on next page)

(suite de la page précédente)

```
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....

2.11 Etape 10 : Etude de l'influence de l'amplitude et de la période pour un signal périodique (version élève)

Notions abordées
- Affichage multiple de graphiques

Référence pyspc
- Les graphiques (deuxième partie)

Consigne et mise en activité :
- Télécharger, faire une copie du notebook « Etude de l'influence de l'amplitude et de la période pour un signal périodique (version élève) » puis répondre aux questions posées.
- Vérifier votre travail si besoin avec le notebook «Etude de l'influence de l'amplitude et de la période pour un signal périodique (version professeur) »

Correction

2.11.1 Etape 10 : Correction : 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 π, 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

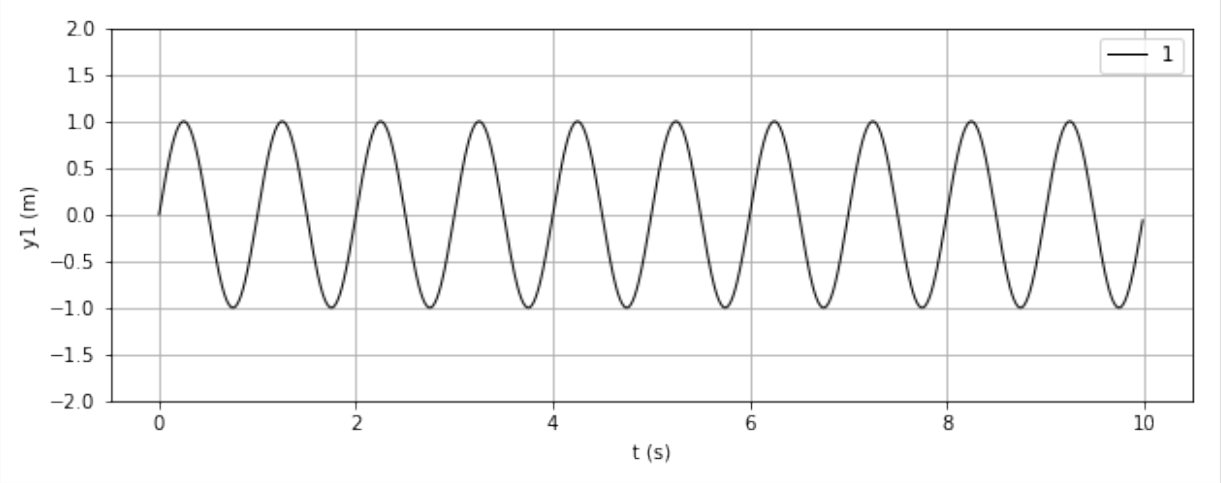
Comme le temps t ne peut pas être continu, il faut le discrétiser, c'est à dire calculer t pour des valeurs entières, multiples d'une petite durée appelée *période d'échantillonnage* et notée Te.

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.

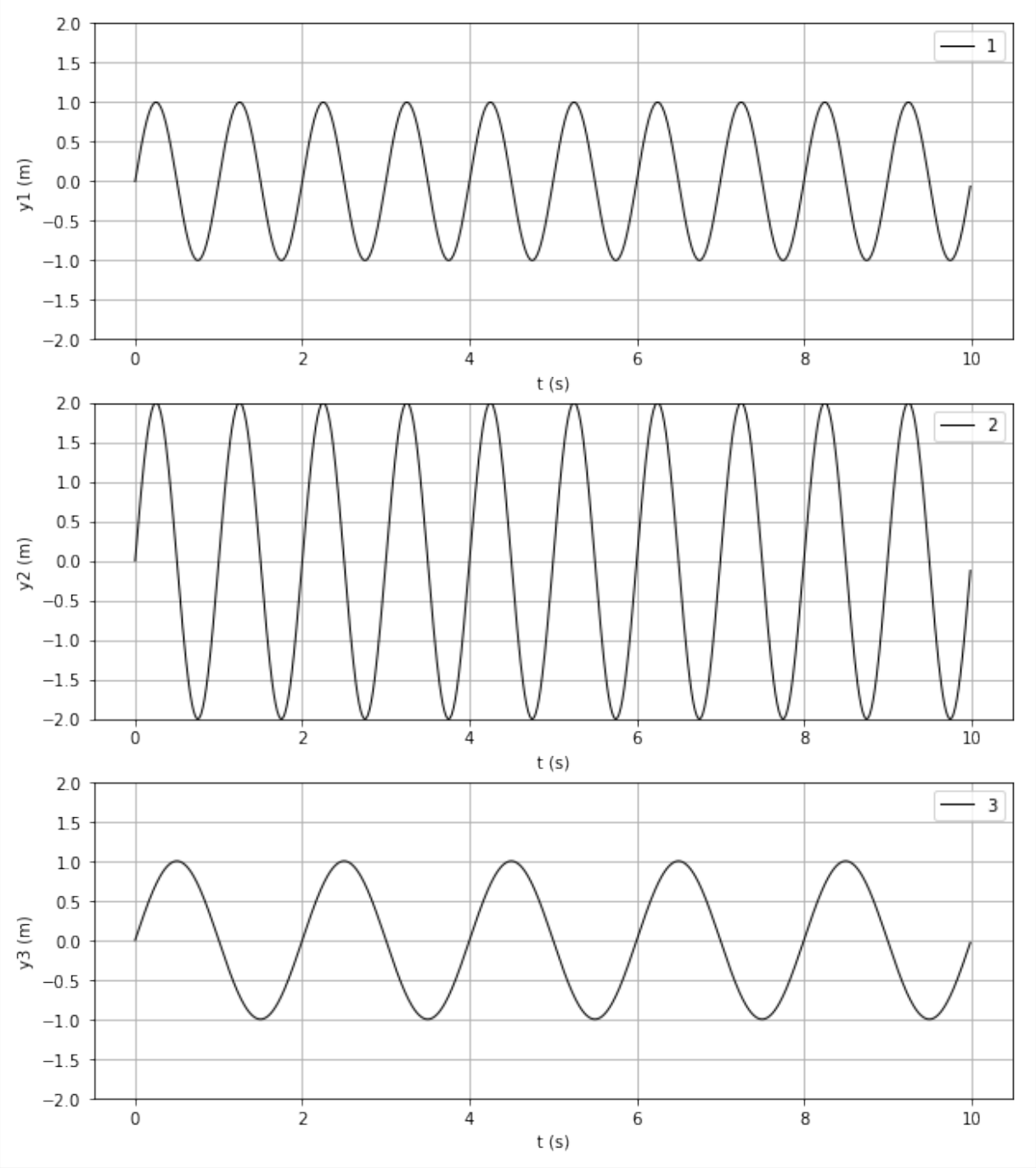
Puis, toujours sur le modèle de la ligne 43, écrire la ligne de code nécessaire à l'affichage de y3.

[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)')
    plt.ylabel(yl)
```

(continues on next page)

(suite de la page précédente)

```
    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()
```



Liens - version élève

`Télécharger le pdf` | `Télécharger le notebook` | `Lancer le notebook sur binder (lent)`

Liens - version prof

`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 π, 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

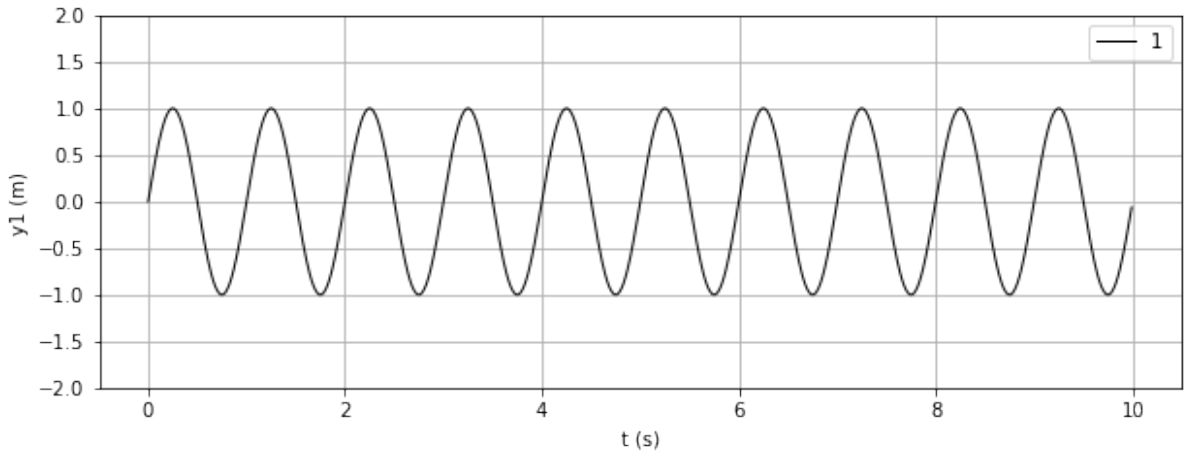
Comme le temps t ne peut pas être continu, il faut le discrétiser, c'est-à-dire calculer t pour des valeurs entières, multiples d'une petite durée appelée *période d'échantillonnage* et notée Te.

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 ?

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.

[ ]:
```
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()
```

2.12 Etape 11 : Mouvement d'un satellite géostationnaire (version élève)

Notions abordées

— Utilisation du langage LaTeX pour écrire des formules dans les cellules Markdown des notebooks

Référence pyspc

— Mémo LaTeX pour les formules de physique

Consigne et mise en activité

— Faire une copie de ce notebook « Mouvement d'un satellite géostationnaire (version élève)» puis répondre aux questions posées.

— Vérifier votre travail si besoin avec le notebook « Mouvement d'un satellite géostationnaire (version professeur)»

Correction

2.12.1 Etape 11 : Correction : Mouvement d'un satellite géostationnaire (version professeur)

`Télécharger le pdf` | `Télécharger le notebook` | `Lancer le notebook sur binder (lent)`

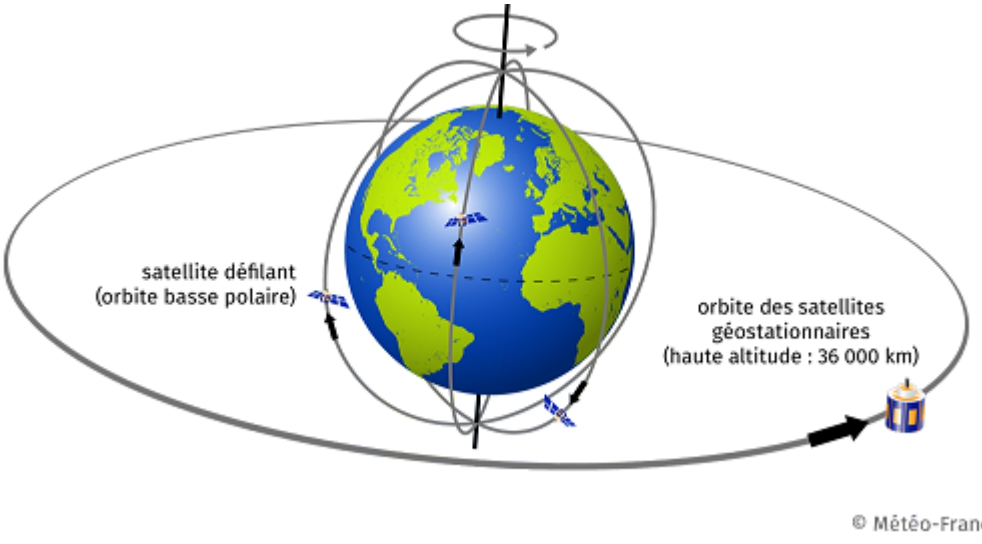
[1]:
```
# cellule 1 : import des bibliothèques

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

Document :

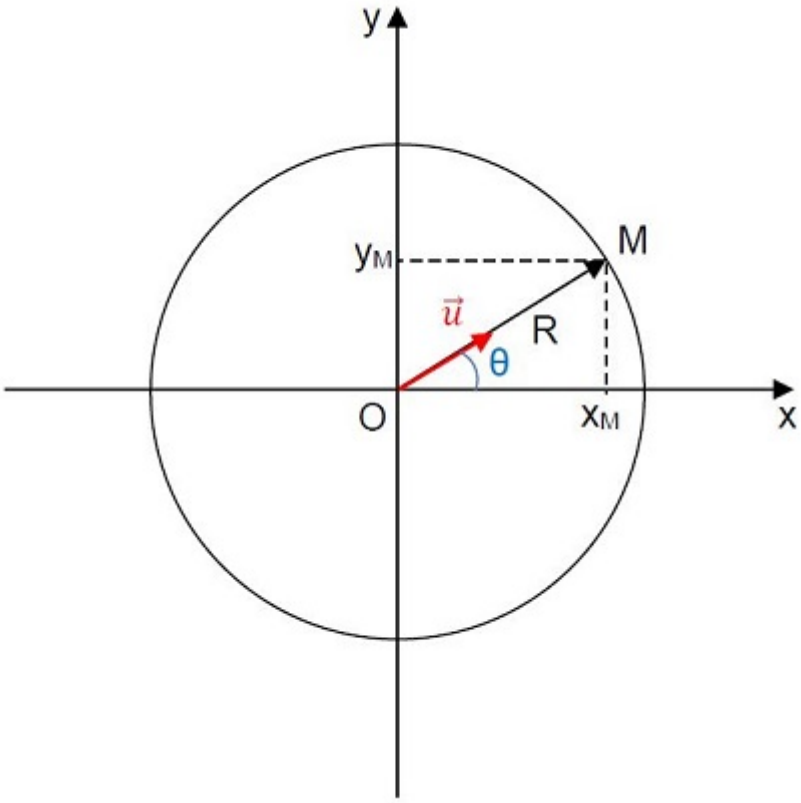
GOES-17 est le deuxième satellite de la génération actuelle de satellites météorologiques exploités par l'Administration nationale des océans et de l'atmosphère (NOAA). Il s'agit d'un satellite géostationnaire qui vise à fournir des images haute résolution visibles et infrarouges et des observations 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).

— 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)

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 θ 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 : θ
— pi : π

5. En déduire l'expression de l'angle θ 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^{eme}' 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 : $\frac{numerateur}{denominateur}$
- 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)
```

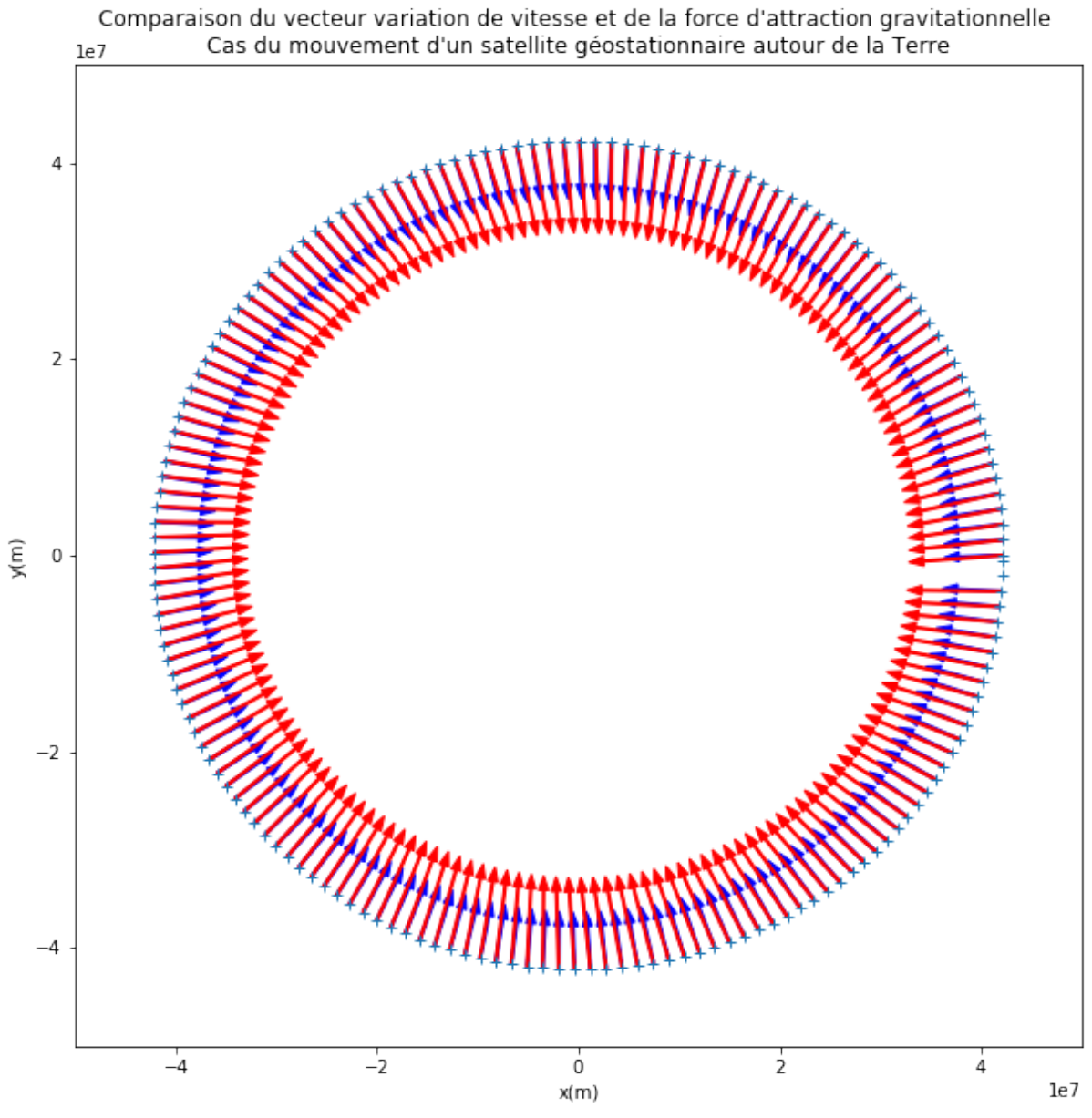
14. Compléter la ligne de code 4 permttant d'afficher la trajectoire du satellite.
15. Compléter les lignes de code 5, 6 et 9 (et éventuellement 10) afin d'afficher les légendes des axes et le titre du graphique.
16. Sur le modèle des lignes de code 13 et 14, compléter les lignes de code 15 et 16 permettant 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)')
plt.ylabel('y(m)')
plt.xlim(-50000000,50000000)
plt.ylim(-50000000,50000000)
```

(continues on next page)

(suite de la page précédente)

```
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+2],10000*Fy[k+2],
              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.

2.12.2 Etape 11 : Correction : Mouvement d'un satellite géostationnaire (version professeur niveau avancé)

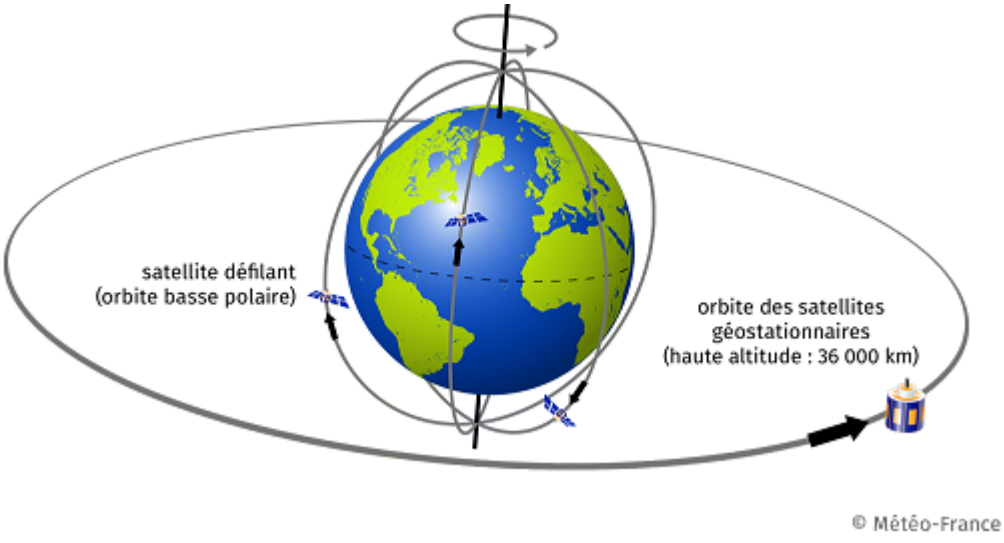
`Télécharger le pdf` | `Télécharger le notebook` | `Lancer le notebook sur binder (lent)`

Document :

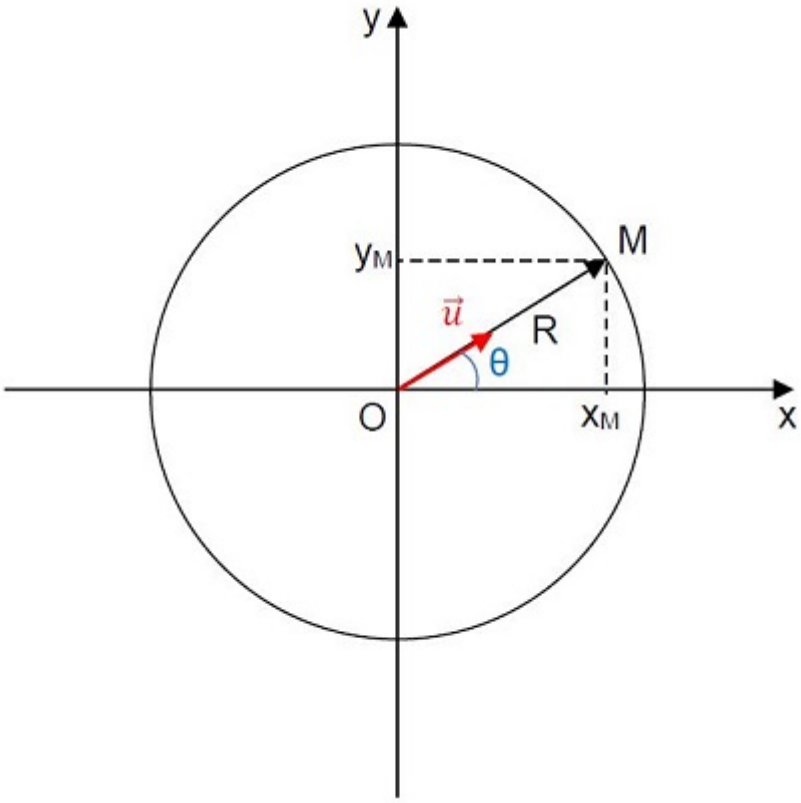
GOES-17 est le deuxième satellite de la génération actuelle de satellites météorologiques exploités par l'Administration nationale des océans et de l'atmosphère (NOAA). Il s'agit d'un satellite géostationnaire qui vise à fournir des images haute résolution visibles et infrarouges et des observations 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 moyenne ? Quel paramètre est-il important de prendre en compte lors de cette étude ?

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
```

4. Quelle valeur en radian prend l'angle θ 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 : θ
— pi : π

5. En déduire l'expression de l'angle θ 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

# Fonction permettant de calculer les coordonnées
# de la position du satellite

def coordposition (t, T) :

    theta=2*np.pi/T*t

    x=R*np.cos(2*np.pi/T*t)
    y=R*np.sin(2*np.pi/T*t)

    return 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).

[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
```

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
```

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 5 et 6 de la cellule 5).

[5]:
```
# cellule 5 : coordonnées du vecteur
# force d'attraction gravitationnelle

def coordforce (x,y,MT,m,R):
    Fx=(G*MT*m/(R**2))*(-x/R)
    Fy=(G*MT*m/(R**2))*(-y/R)
    return Fx,Fy
```

14. Compléter la ligne de code 7 permttant d'afficher la trajectoire du satellite.
15. Compléter les lignes de code 10, 11 afin d'afficher les légendes des axes.
16. Sur le modèle des lignes de code 19, 20 et 21 , compléter les lignes de code 22, 23 et 24 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 en fonction de la durée Te entre
# deux positions successives du satellite.

def trace (x,y,dvx,dvy,Fx,Fy,Te,n) :
    plt.subplot(1,3,n)
    plt.plot(x,y,'k+')
    plt.plot(0,0,'go')
    plt.text(1000000,1000000,"Terre")
    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.text(20000000,40000000,"Te="+ str(Te) +" jours \n")

    for k in range(len(dvx)):
        plt.arrow(x[k],y[k],30000*dvx[k],30000*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)
```

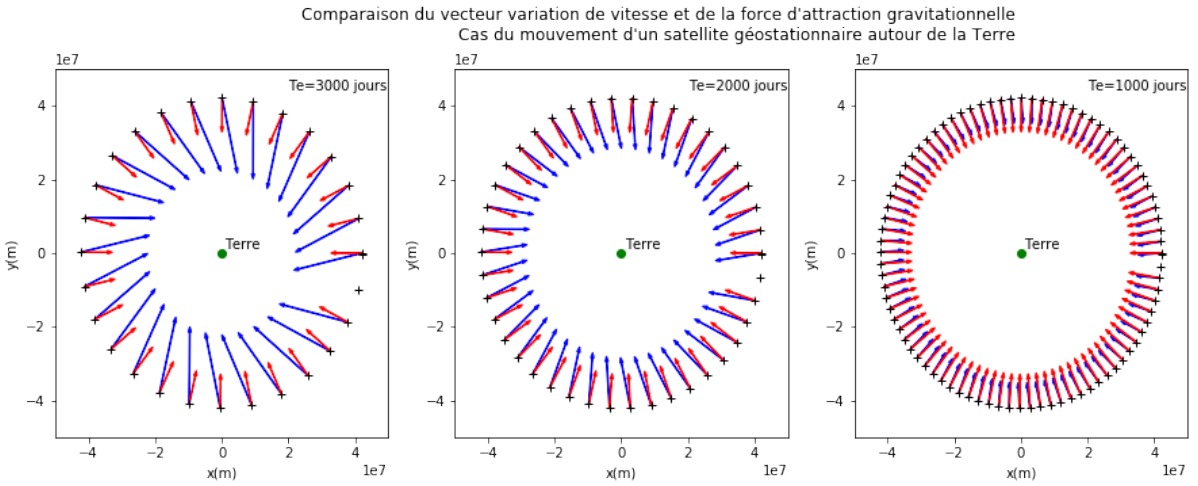
3. A l'aide du document, déterminer les valeurs du rayon R de la trajectoire et la période de révolution T du satellite (lignes 3 et 4 de la cellule 7).

[7]:
```
# cellule 7 : Programme principal permettant d'afficher
# 3 graphiques avec des durées entre deux positions successives
# du satellite différentes

R=42164000   # rayon en m
T=84164      # période de révolution en s

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

plt.figure (figsize=(15,5))


# Graphique 1
Te1=3000      # durée entre deux positions successives en jours
t1=np.arange(0,T,Te1)
x1,y1=coordposition (t1,T)
vx1=coordvit(t1,x1)
vy1=coordvit(t1,y1)
dvx1=coordvarvit(vx1)
dvy1=coordvarvit(vy1)
Fx1,Fy1=coordforce(x1,y1,MT,m,R)
trace(x1,y1,dvx1,dvy1,Fx1,Fy1,Te1,1)

# Graphique 2
Te2=2000      # durée entre deux positions successives en jours
t2=np.arange(0,T,Te2)
x2,y2=coordposition (t2,T)
vx2=coordvit(t2,x2)
vy2=coordvit(t2,y2)
dvx2=coordvarvit(vx2)
dvy2=coordvarvit(vy2)
Fx2,Fy2=coordforce(x2,y2,MT,m,R)
trace(x2,y2,dvx2,dvy2,Fx2,Fy2,Te2,2)

# Graphique 3
Te3=1000      # durée entre deux positions successives en jours
t3=np.arange(0,T,Te3)
x3,y3=coordposition (t3,T)
vx3=coordvit(t3,x3)
vy3=coordvit(t3,y3)
dvx3=coordvarvit(vx3)
```

(continues on next page)

(suite de la page précédente)

```
dvy3=coordvarvit(vy3)
Fx3,Fy3=coordforce(x3,y3,MT,m,R)
trace(x3,y3,dvx3,dvy3,Fx3,Fy3,Te3,3)

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 \n",horizontalalignment='right')

plt.show()
```



17. Conclusion : Répondre à la problématique dans la cellule suivante en double-cliquant dessus si besoin.

Liens version élève

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

Liens version prof

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

Liens version prof niveau «avancé»

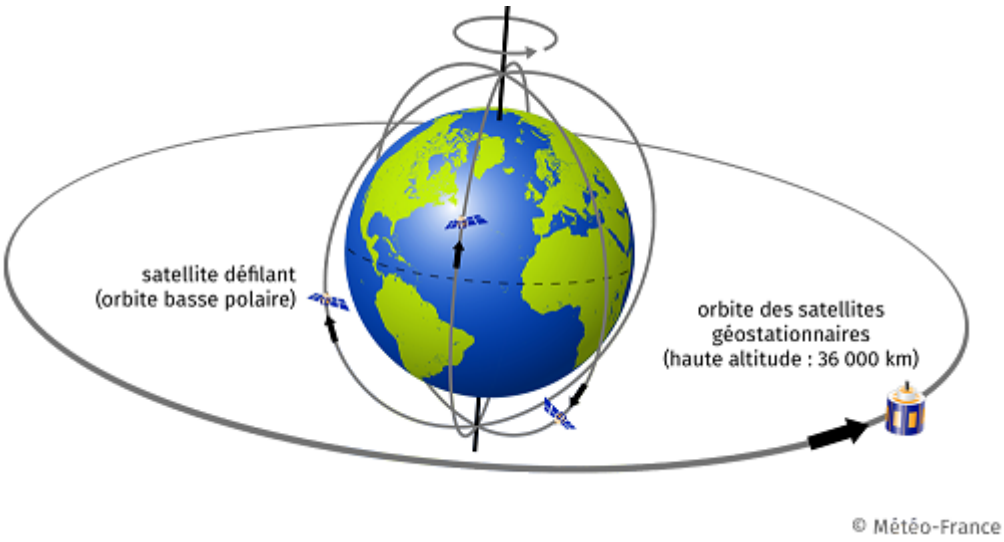
Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

Document :

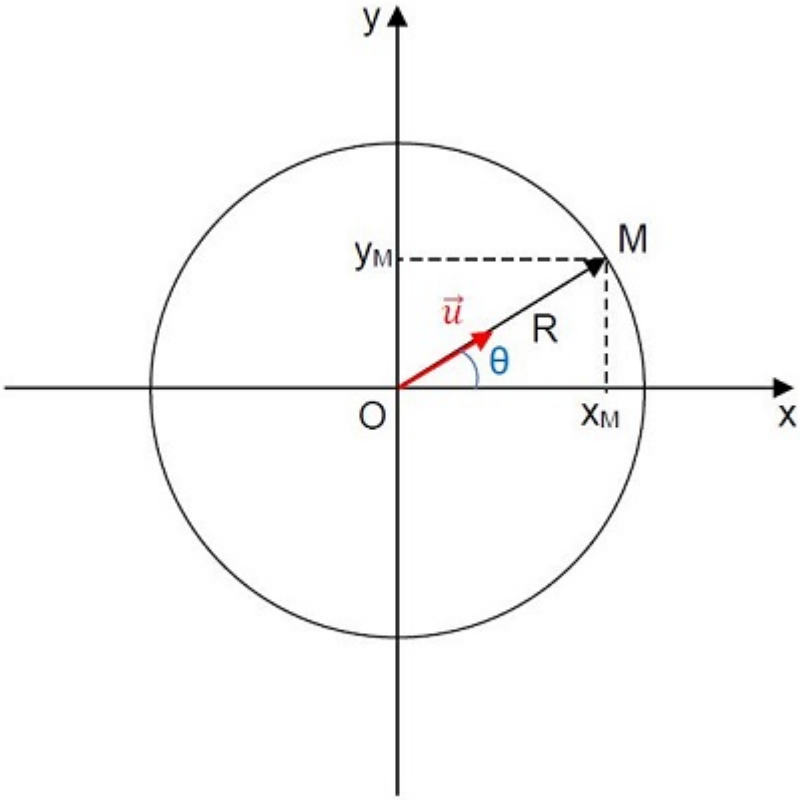
GOES-17 est le deuxième satellite de la génération actuelle de satellites météorologiques exploités par l'Administration nationale des océans et de l'atmosphère (NOAA). Il s'agit d'un satellite géostationnaire qui vise à fournir des images haute résolution visibles et infrarouges et des observations 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 θ 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 : θ

— pi : π

5. En déduire l'expression de l'angle θ 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 $\vec{u}$. (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.

2.13 Etape 12 : Animation d'une onde le long d'une corde

Ce notebook peut être est long à charger, patientez :)

Notions abordées
— Animation d'un graphique
Référence pyspc
— Les animations de graphiques
Consigne
— Etudier ce programme

`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)
```

(continues on next page)

(suite de la page précédente)

```
    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
```

[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 0x7f470f987fd0>
```

[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
# 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...
```

[3]:
```
<matplotlib.animation.FuncAnimation at 0x7f470f63c390>
```

2.14 Etape 13 : Quelques exemples et bases de travail

2.14.1 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 # ou import ipywidgets as widgets
                   #    %matplotlib widget
```

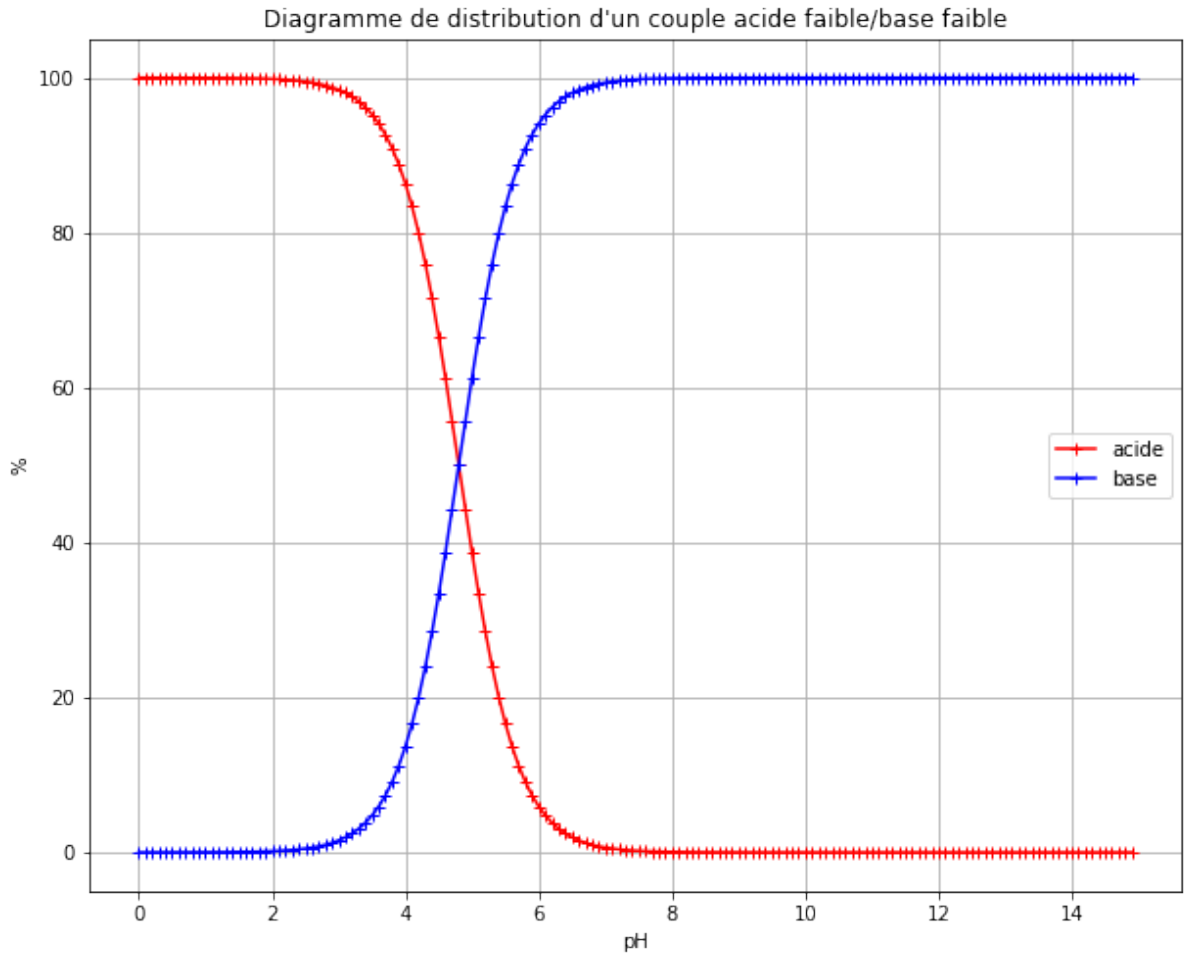
(continues on next page)

(suite de la page précédente)

```
                    # pour avoir accès au zoom et au pointeur

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)
```



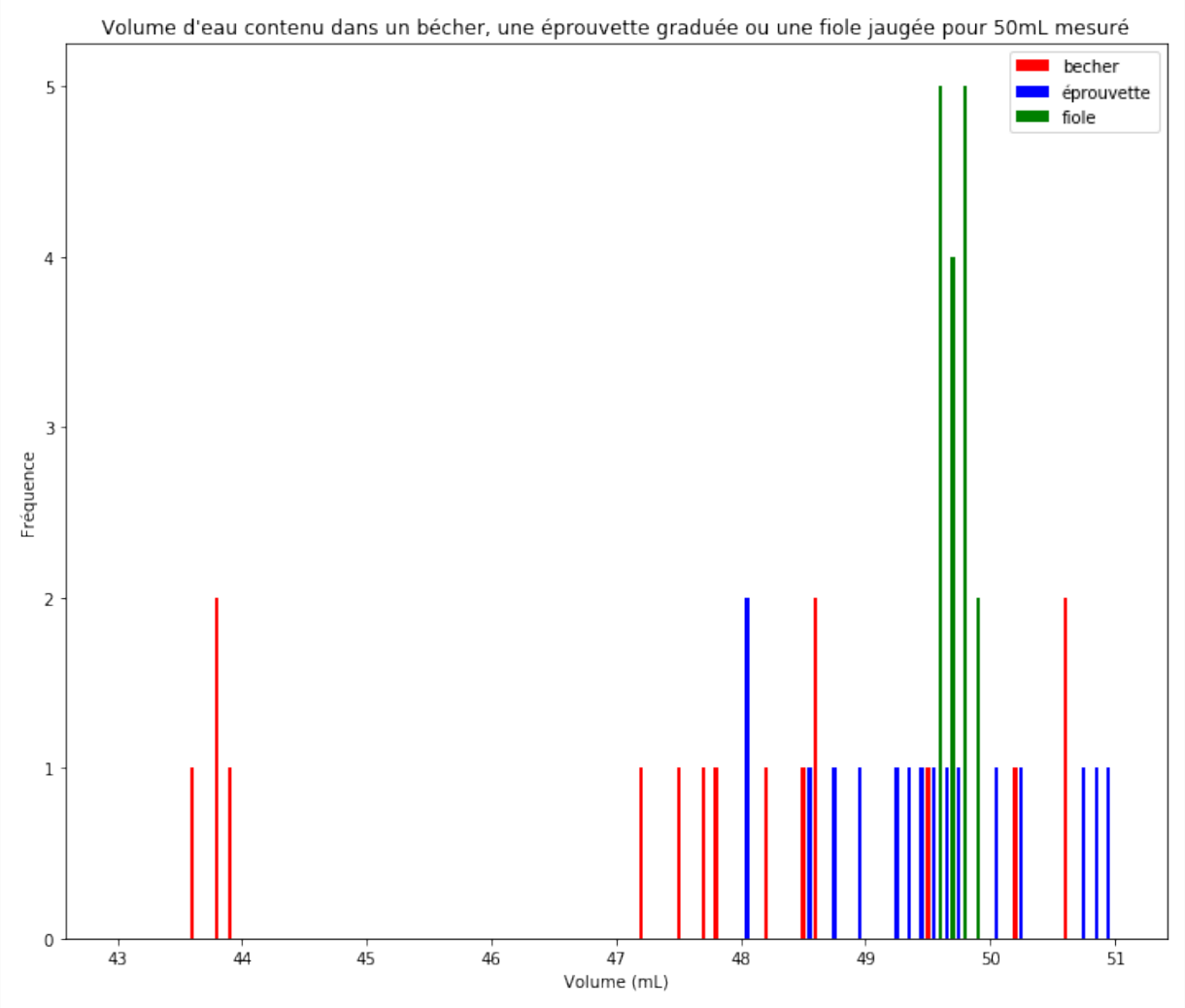
[ ]:

2.14.2 Histogramme et évaluation de l'incertitude-type de type A sur une série de mesures

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

[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.
↪64,49.74,49.81,49.50,49.59]
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")
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)
```

[ ]:

2.14.3 Interférences

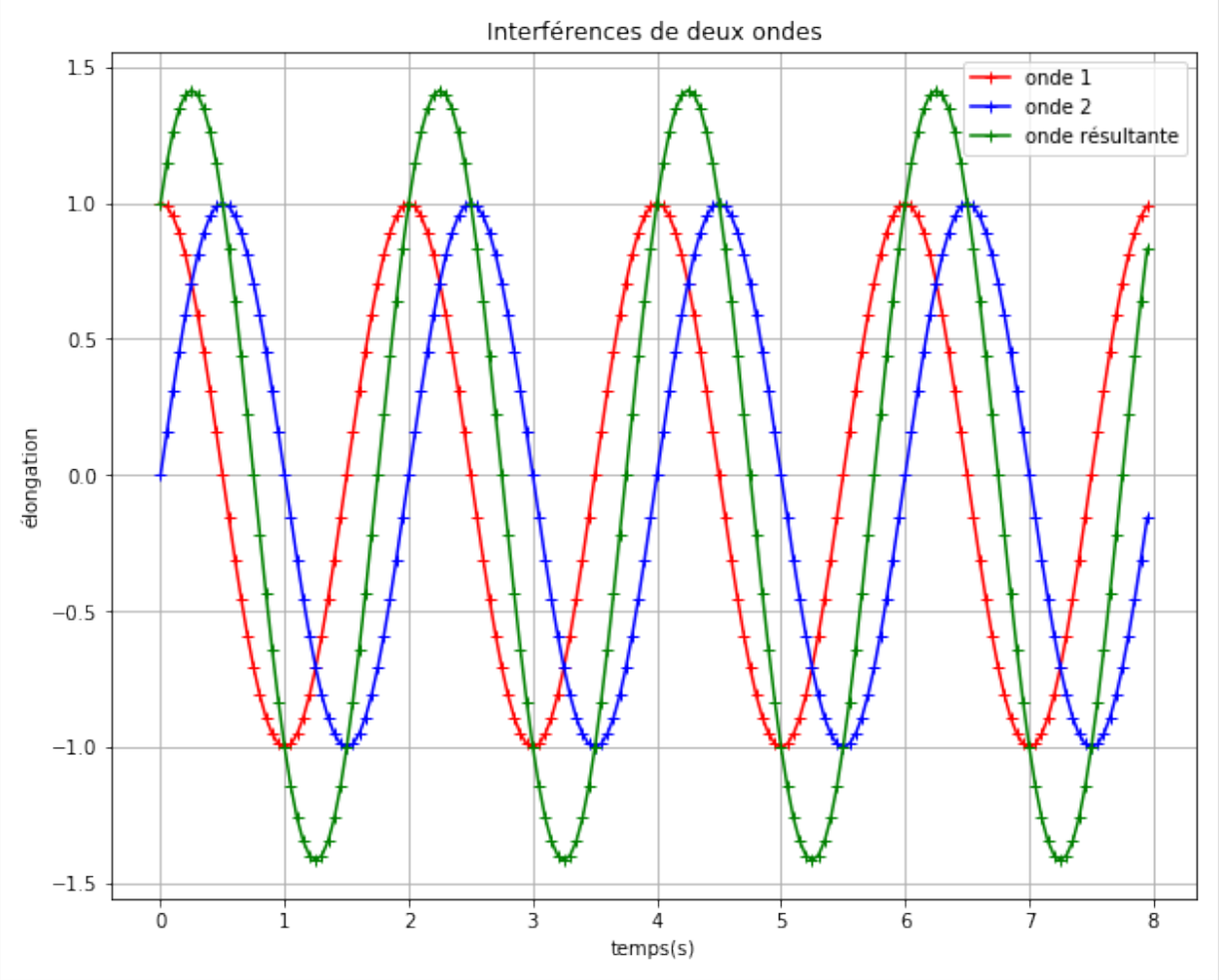
2.14.3.1 Somme de deux signaux sinusoïdaux synchrones

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

[1]:
```
# Import des bibliothèques

import numpy as np
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)
```



2.14.4 Réactions acido-basiques

2.14.4.1 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)
```

(continues on next page)

(suite de la page précédente)

```
    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, l'avancement maximal vaut",xmax,
↪"mol \n "
           "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
    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)
```

[ ]:
```

```

2.14.5 Titrage suivi par pH-métrie

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

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

[1]:
```
# Import des bibliothèques

import matplotlib.pyplot as plt

%matplotlib inline
# ou import ipywidgets as widgets
#    %matplotlib widget
# pour avoir accès au zoom et au pointeur

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]
```

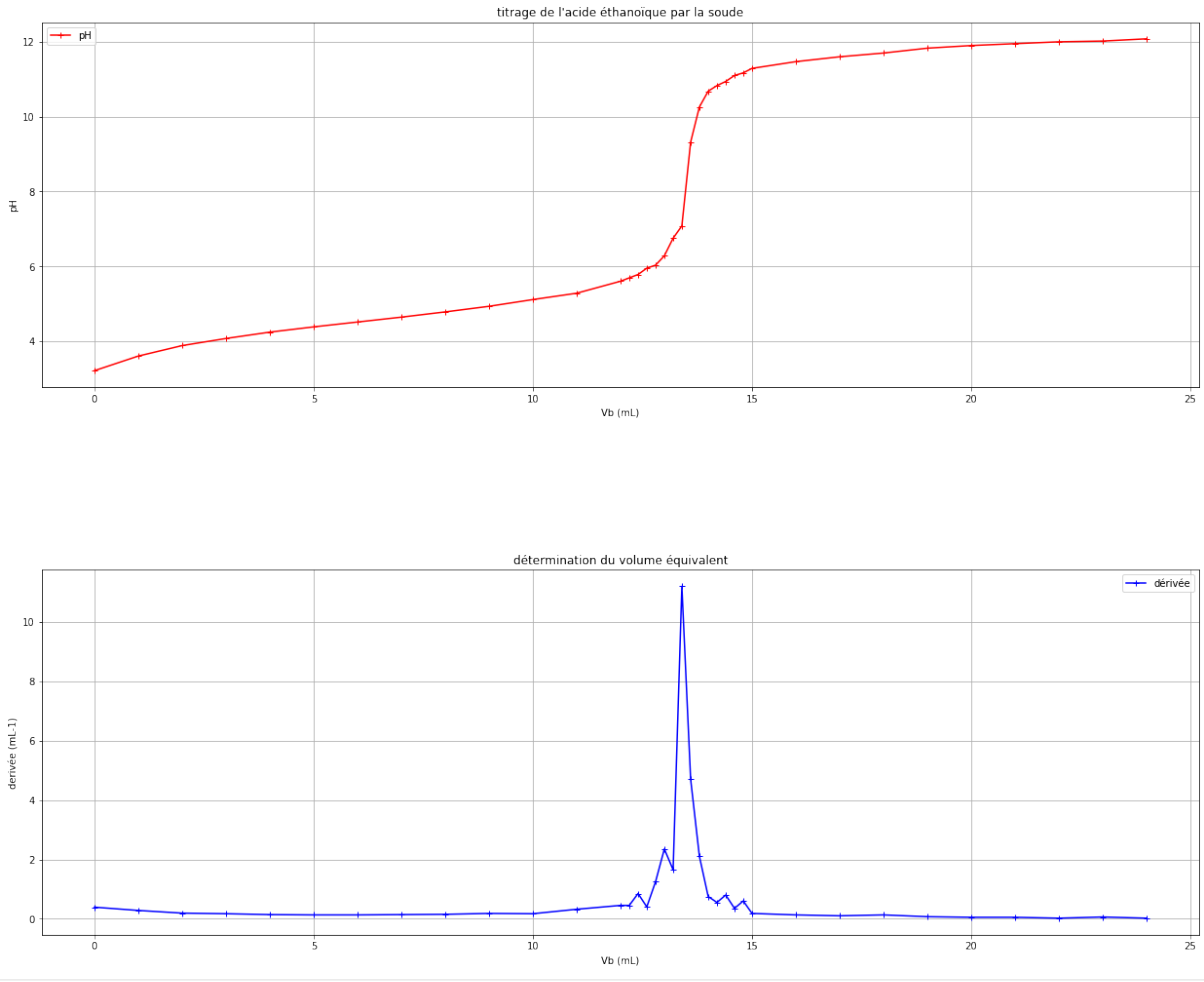
[5]:
```
# Suppression de la dernière valeur du tableau à cause de l'affichage de la courbe␣
↪de la dérivée
Vb = np.delete(Vb,-1)
pH = np.delete(pH,-1)
```

[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)")
```

(continues on next page)

(suite de la page précédente)

```
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)):
```

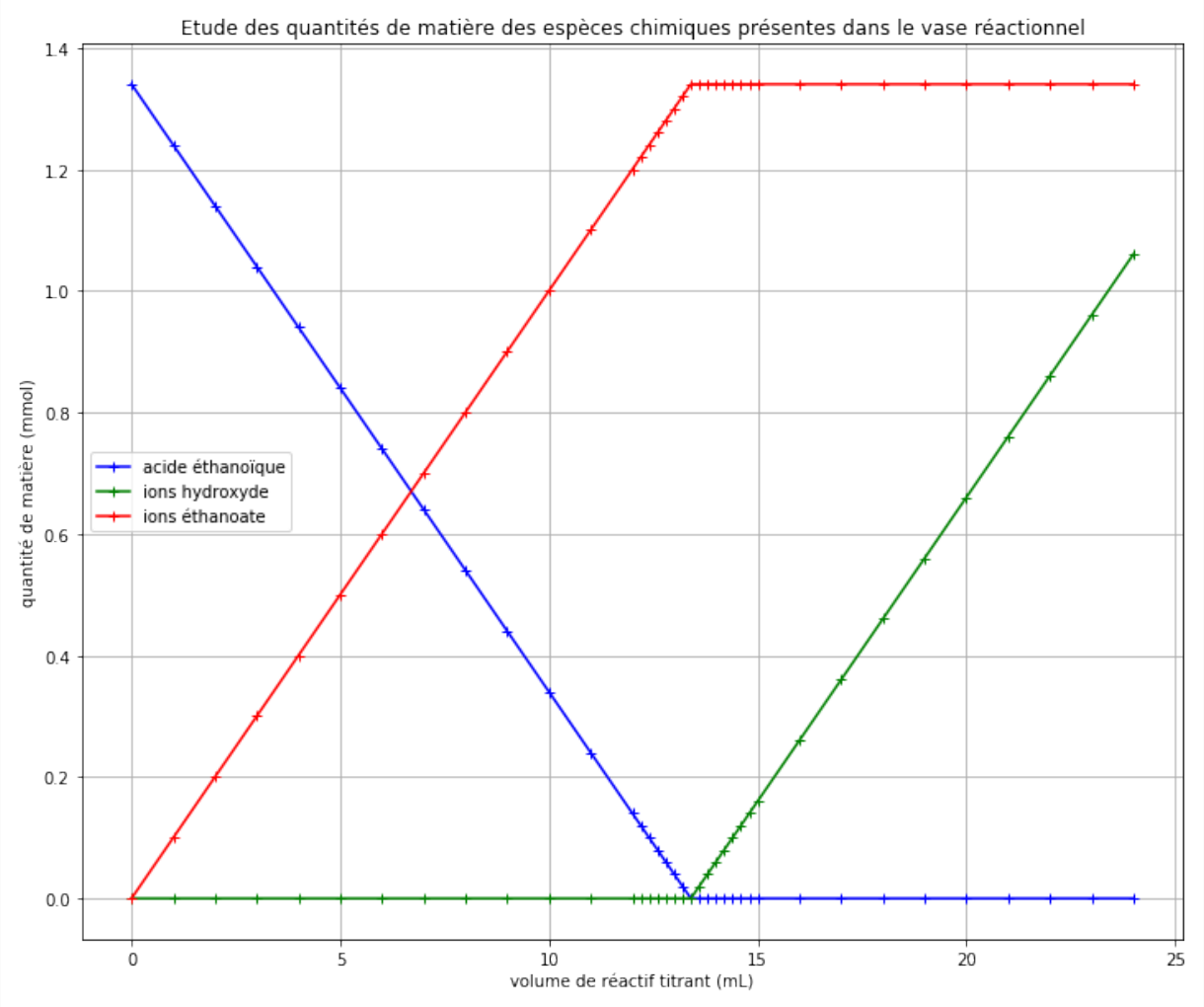
(continues on next page)

(suite de la page précédente)

```
    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)
        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.   0.02 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()
```



[ ]:

2.15 Réinvestissement J1

Télécharger le pdf | Télécharger le notebook | Lancer le notebook sur binder (lent)

2.15.1 Programme 1

2.15.1.1 Niveau 1

Ecrire un programme qui, à partir de la distance focale d'une lentille convergente et la position d'un objet par rapport à cette lentille, affiche la position de l'image.

Ajouter un message au début précisant que les distances s'expriment en mètre.

Faire un test qui affiche « impossible » si le l'objet se trouve sur le foyer objet.

Prévoir l'affichage d'un message qui précisera si l'image est réelle ou virtuelle.

2.15.1.2 Niveau 2 (activité 6 atteinte)

Reprendre ce programme mais en définissant une fonction position_image qui renvoie la position de l'image ou None si impossible, et intégrer cette fonction dans un programme principal sur quelques exemples.

2.15.2 Programme 2

2.15.2.1 Application de la loi de Beer Lambert

En reprenant l'activité sur la loi d'Ohm, faire un programme affichant :
- le graphique de la courbe $A = f(c)$
- la droite de régression en utilisant la fonction polyfit
- l'équation de la droite

[ ]:

2.16 Accès et téléchargements

CHAPITRE 3

Accès

Ce contenu de formation est accessible parallèlement au guide Python pour la SPC au lycée :

- 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)
- sur gitlab

CHAPITRE 4

Licence

Auteurs, par ordre alphabétique

- BARBIER Jean-Matthieu <jean-matthieu.barbier@ac-rouen.fr>
- DENIS Cyril <cyril.denis@ac-rouen.fr>
- DEVEDEUX Dominique <dominique.devedeux@ac-rouen.fr>
- DENDIEVEL Alexis <alexis.dendievel@ac-rouen.fr>
- REBOLINI Gaelle <gaelle-nathalie.rebolini@ac-rouen.fr>