Introduction à Python

Pour commencer, connectez-vous au serveur L2 de la plateforme Jupyter de Lyon 1 : https://l2-nbgrader.univ-lyon1.fr avec vos identifiant et mot de passe de l'université.

Récupérer le notebook de ce TP en procédant comme suit :

1. Choisir *Nbgrader –> Assigment List*.
2. Dans le menu *Released, downloaded, and submitted assignments for course*, choisir MAT2092L.
3. Puis *Fetch* `TP0_2024` dans la rubrique *Released assignments*.
4. Cliquer sur le lien `TP0_2024` se trouvant à gauche dans la rubrique *Downloaded assignments*, puis juste en-dessous sur le lien vers le notebook `TP0_2024_analyse3`

Ce TP à réaliser en autonomie est particulièrement recommandé pour les étudiants et étudiantes n'ayant pas suivi l'an dernier l'UE d'analyse 2 à Lyon 1. Il est constitué en grande partie du TP1 d'analyse 2.

Pour commencer, voici quelques manipulations pour prendre en main python. Pour voir le résultat des commandes indiquées dans une cellule, il faut l'exécuter, en appuyant sur la flèche ▶ en haut ou en appuyant sur "Maj+Entrée".

```
1+1
```

```
# 1+1
# quand on veut écrire quelque chose qui ne␣
 ↪sera pas exécuté, on met un # devant...
1+2
```

```
print('hello')
```

Pour utiliser l'aide de python au sujet d'une commande, on peut taper `help(commande)`.

```
help(round)
```

Que fait la commande `round` ? Exécuter la cellule suivante pour confirmer.

```
#
#
#
```

```
round(1.43267,3)
```

Opérations

On peut facilement faire les opération usuelles.

```
2+5
```

```
2345/34578
```

```
2*3
```

```
2**3
```

```
5**(1/2)
```

Que fait `**`?

```
#
#
```

Attention ! Si on met plusieurs calculs à la suite sans commande `print`, seul le résultat du dernier calcul est affiché, comme dans la cellule suivante.

```
2+5
2345/34578
2*3
2**3
5**(1/2)
```

Définition de variables

```
a = 10
b = a + 5
b
```

```
print('a =',a,"et b =",b)
# entre '.' ou "." on met les chaînes de␣
 ↪caractères, elles sont affichées telles␣
 ↪qu'elles sont écrites
```

```
a, b, c = 1, 2, 3
print(a, b, c)
```

```
a, b = b, a
print('a =',a,'et b =',b)
```

```
s = 'Hello !'
print(s)
```

Les fonctions usuelles

Nous utiliserons le package ***Numpy*** qui est le package de référence pour le calcul numérique en Python. En

particulier, toutes les fonctions usuelles existent dans ce package. Et aussi des nombres comme e, π ...

Pour utiliser un package, il faut l'importer. Pour cela on utilise la commande suivante. Ce sera souvent la première ligne écrite dans un notebook Jupyter.

```
import numpy as np
```

Voici des exemples. Identifier les fonctions et les nombres ci-dessous.

```
x1 = np.abs(-2)
x2 = np.sqrt(3)/2
x3 = np.exp(1)
x4 = np.log(np.e)
x5 = np.sin(np.pi/6)
x6 = np.sin(np.pi/3)
print(x1, x2, x3, x4, x5, x6)
```

```
#
#
#
```

Définir une fonction dans Python

Une fonction Python se déclare comme suit:

où les `input` sont les arguments d'entrée de la fonction et les `output` sont les arguments de sortie. Ni les uns ni les autres ne sont obligatoires.

Attention ! Les indentations sont très importantes en Python. C'est ce qui définit si les instructions font partie de la fonction ou du script. En effet, il n'y a rien qui indique la fin de la fonction à part les indentations !

Par exemple, pour coder la fonction $f : x \mapsto \frac{1}{(1+x)^2} - x$, et l'évaluer en 7, on écrit les lignes suivantes.

```
def f(x):
    return 1/(x+1)**2 - x

print(f(7))
```

Définir la fonction $f : x \mapsto e^{\sqrt{x+1}}$, et l'évaluer en 1.

```
# à compléter
```

Boucles et tests

Nous allons maintenant voir comment effectuer des tests et des boucles.

Commençons par les tests avec le mot-clé `if`. La syntaxe est la suivante :

- Les symboles `:` après `if`, `elif` et `else` sont obligatoires. Ils marquent le début des blocs.
- Les indentations sont importantes ! Elles définissent si les instructions font partie des blocs ou non. Il n'y a, en effet, pas de mot-clé pour marquer la fin des blocs en Python.
- Les mot-clés `elif` et `else` ne sont pas obligatoires.

Voici un exemple.

```
def f(x):
    if x < 0:
        return -x
    elif x == 0 or x == 1:
        return 1-x
    else:
        return x**2
```

```
print(f(-5), f(1), f(3), f(1.5))
```

Les boucles `for`

La syntaxe est :

- La variable `variable_boucle` va, une par une, prendre les valeurs dans la variable `une_variable`. Pour chacune de ces valeurs, les instructions dans le bloc `for` seront exécutées.
- `une_variable` peut être, par exemple, une liste Python ou un tableau ; le plus souvent ce sera un intervalle d'entiers.
- Comme pour les `if`, les `:` et les tabulations sont nécessaires.

```
for i in range(5): # range(5) est␣
↪l'intervalle des entiers compris entre 0␣
↪et 4
    print(i*i)
```

```
# range(p,q) est l'intervalle des entiers␣
↪compris entre p et q-1
u=1
for i in range(1,7):
    u=2*u
    print(u)
```

Les boucles `while`

La syntaxe est:

Les instructions dans le bloc `while` sont exécutées tant que `condition` est vraie. Voici un exemple.

```
n=0
u=(2*n+1)/(n+1)-2
while np.abs(u) > 10**(-4):
    n=n+1
    u=(2*n+1)/(n+1)-2

print(n)
```

Qu'a-t-on calculé à la cellule précédente ?

```
#
#
#
#
#
```

Les tableaux avec Numpy

On utilise souvent des tableaux pour faire des calculs avec Python : les tableaux peuvent être des vecteurs,

des matrices...

Dans ces TP d'analyse 2, les tableaux que nous utiliserons seront des vecteurs, des "tableaux à une dimension".

Voici des exemples. Observer en particulier que les opérations sont appliquées élément par élément.

```
u = np.array([1, -2, 7, 0, 10])
v = np.array([3, -1, 3, 4, 1])
print(u)
print(2 * u)
print(u + v)
print(u * v)
```

Les éléments des tableaux sont numérotés à partir de 0. Tester les différentes commandes suivantes permettant d'extraire un ou plusieurs éléments d'un tableau.

```
print(u[0], u[1])
print('le dernier élément de u est', u[-1])
```

```
print(u[0:4]) # remarquer que u[p:q]␣
↪renvoie les éléments de u de l'indice p à␣
↪l'indice q-1
print(u[1:3])
```

Écrire une commande permettant d'extraire les éléments de u de l'indice 2 à l'indice 4.

```
# à compléter
```

```
u[0:6:2] # on peut extraire les termes␣
↪d'indices pairs de u de cette façon
```

Écrire une commande permettant d'extraire les termes d'indices impairs de u.

```
# à compléter
```

Création automatique de tableaux structurés

Observer le résultat de chaque commande pour comprendre son fonctionnement, en allant éventuellement regarder dans l'aide.

```
print(np.arange(0,10))
print(np.arange(0,10,0.5))
```

Quels sont les éléments du tableau créé par la commande `np.arange(a,b,h)` ?

```
#
#
#
#
#
```

```
print(np.linspace(0, 10, 10))
print(np.linspace(0, 10, 11))
print(np.linspace(0, 10, 21))
print(np.linspace(0,2*np.pi,7))
```

Quels sont les éléments du tableau créé par la commande `np.linspace(a,b,n)` ?

```
#
#
#
#
#
```

Il est également possible de créer un tableau à partir d'une formule et d'une boucle `for`.

```
x = np.array([i**2 for i in range(10)])
print('x=', x)

y = np.array([np.cos(2*k*np.pi/6) for k in␣
↪range(7)])
print('y=', np.round(y,3))
```

On peut calculer le maximum et le minimum des éléments d'un tableau.

```
x = np.array([2,-1,np.pi,-3,-2,-np.e])
print('x=',x)
print('le plus grand élément de x est', np.
↪max(x))
print('le plus petit élément de x est', np.
↪min(x))
```

Et aussi demander la taille d'un tableau.

```
print("le nombre d'éléments de x est", np.
↪size(x))
```

Les fonctions prédéfinies dans Numpy s'appliquent à des tableaux en opérant élément par élément.

```
np.sqrt(np.abs(x))
```

Exercice 1 - Approximations du maximum d'une fonction

Définir une fonction **max_approx** prenant en argument une fonction réelle f, les extrémités a et b et un entier $n \geq 1$, et retournant la valeur

$$\max_{0\leq k\leq n} f\left(a + k\frac{(b-a)}{n}\right).$$

```
import numpy as np
def max_approx(f,a,b,n):
    # compléter - utiliser la commande␣
↪linspace
    return 0 # à modifier
```

```
#Test avec la fonction sinus sur [0,2]
print([max_approx(np.sin,0,2,10**i) for i␣
↪in range(2,7)])
```

En TD (voir fiche 1), on a montré que pour tout $n \geq 1$,

$$\left|\max_{0\leq k\leq n} \sin\left(\frac{2k}{n}\right) - 1\right| \leq \frac{1}{2n^2}.$$

Observer ce que retourne la commande suivante :

```
#Majoration de l'erreur
print([(1-max_approx(np.
 ↪sin,0,2,10**i))*(100**i) for i in␣
 ↪range(2,7)])
```

Coder la fonction $f : \mathbb{R} \to \mathbb{R}$, $|x \mapsto x(x-1)|$. Puis estimer son maximum sur $[0, 1]$.

```
#
#
#
#
#
```

Coder la fonction $g : \mathbb{R} \to \mathbb{R}$, $x \mapsto |(x+1)x(x-1)|$. Puis estimer son maximum sur $[-1, 1]$.

```
#
#
#
#
#
```

Tracer des graphiques

Il existe beaucoup de packages en Python pour tracer des graphiques. Le plus utilisé est `matplotlib`. Pour charger les fonctions principales de ce package, on utilise la commande suivante :

```
import matplotlib.pyplot as plt
```

Nuage de points et points reliés

Pour représenter avec Python un nuage de points $\{(x_k, y_k) : 1 \leq k \leq N\}$, on définit un vecteur contenant les abscisses $(x_1, \ldots, x_N)$ et un vecteur contenant les ordonnées $(y_1, \ldots, y_N)$ des points dans le même ordre. On ajoute ensuite le marqueur souhaité pour matérialiser les points, par exemple une croix `+` dans la couleur souhaitée `r`.

```
N = 20
x = np.array([np.cos(2*i*np.pi/N) for i in␣
 ↪range(N+1)])
y = np.array([np.sin(2*i*np.pi/N) for i in␣
 ↪range(N+1)])
plt.plot(x, y, 'r+')
```

Pour relier les points, il suffit d'ajouter le type de trait par lequel on souhaite que les points soient reliés : `-` pour un trait continu, ou `--` pour un trait discontinu par exemple. Tester les deux propositions successivement.

```
N = 20
x = np.array([np.cos(2*i*np.pi/N) for i in␣
 ↪range(N+1)])
y = np.array([np.sin(2*i*np.pi/N) for i in␣
 ↪range(N+1)])
plt.plot(x, y, 'r+-')
#plt.plot(x, y, 'm*--')
plt.axis('equal') # pour avoir un repère␣
 ↪orthonormé (on voit mieux que les points␣
 ↪sont sur un cercle...)
```

```
N = 20
x = np.array([np.cos(2*i*np.pi/N) for i in␣
 ↪range(N+1)])
y = np.array([np.sin(2*i*np.pi/N) for i in␣
 ↪range(N+1)])
plt.plot(x, y, 'm--')
plt.plot(x, y, 'g*')
plt.axis('equal') # pour avoir un repère␣
 ↪orthonormé
```

Représentation des termes d'une suite

Pour visualiser les N premiers termes d'une suite $(u_n)_{n\in\mathbb{N}}$, on peut dessiner le nuage de points $\{(n, u_n) : 0 \leq n \leq N\}$ (réliés ou non). Pour cela, on commence par définir le vecteur des abscisses $(0, \ldots, N)$ puis le vecteur des ordonnées $(u_0, \ldots, u_N)$. On peut finalement tracer la suite en précisant que les marqueurs sont par exemple des ronds `o` bleus `b`.

Dans l'exemple qui suit, la suite $(u_n)_{n\in\mathbb{N}}$ est définie par $u_n = n^2 + 1$ pour tout $n \in \mathbb{N}$.

```
n = np.arange(25)

def u(n):
    return # à compléter

plt.plot(n,u(n), 'bo')
```

Titres et légendes

Toute figure doit être accompagnée d'une légende, et d'un titre pour chacun des axes et pour la figure.

```
plt.plot([0, 1], [1, 0], 'b',␣
 ↪label='Segment')
plt.plot(0.1, 0.9, 'r+', label='Point')
plt.xlabel('en abscisse')
plt.ylabel('en ordonnée')
plt.title('Le titre de la figure')
plt.legend()
```

Représentation graphique de fonctions avec Python

Nous allons tracer des courbes représentatives de fonctions réelles d'une variable réelle.

Pour faire de tels graphiques, on utilise le même package que celui utilisé plus haut `matplotlib`.

Pour tracer le graphe d'une fonction $f : D_f \to \mathbb{R}$, on choisit d'abord un segment $[a, b]$ inclus dans D_f, où vont être prises les abscisses. On ne peut pas demander à Python de calculer une infinité de valeurs, on va donc créer un vecteur d'abscisses contenant "beaucoup" de

points dans $[a, b]$ (par exemple une centaine) :

$$(x_0, \ldots, x_N) \quad \text{avec} \quad a = x_0 < x_1 < \ldots < x_N = b$$

Ensuite, on va créer le vecteur des ordonnées correspondantes $(f(x_0), \ldots, f(x_N))$.

La commande la plus simple possible qui permet de faire un tracé est : `plt.plot(x,y)`, où `x` est le tableau contenant les abscisses et `y` le tableau contenant les ordonnées.

Essayons d'abord avec la fonction sin, sur l'intervalle $[0, 2\pi]$.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 100) # le␣
 ↪vecteur des abscisses
plt.plot(x, np.sin(x))
```

```
x = np.linspace(0, 2*np.pi, 10) # si on ne␣
 ↪met pas assez de points en abscisses on␣
 ↪voit bien que Python relie
# les points par des droites, ce n'est pas␣
 ↪très joli...
plt.plot(x, np.sin(x))
```

On peut aussi tracer deux courbes sur la même figure.

```
x = np.linspace(0, 2*np.pi, 100) # le␣
 ↪vecteur des abscisses
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
```

Quand on dessine, il est préférable d'obtenir une représentation plus lisible. On peut ajouter des options pour faire apparaître les axes des abscisses et des ordonnées par exemple.

```
x = np.linspace(0, 2*np.pi, 100) # le␣
 ↪vecteur des abscisses

# ces quelques lignes, facultatives,␣
 ↪permettent de faire apparaître les axes␣
 ↪habituels des abscisses et des ordonnées
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
# fin des lignes facultatives

plt.plot(x, np.sin(x), label='sinus')
plt.plot(x, np.cos(x), label='cosinus')
plt.xlabel('x') # pour donner un nom à␣
 ↪l'axe des abscisses
plt.ylabel('y') # pour donner un nom à␣
 ↪l'axe des ordonnées
plt.title('joli graphe') # pour donner un␣
 ↪nom à la figure
plt.legend() # pour indiquer une légende et␣
 ↪donc savoir quelle courbe correspond à␣
 ↪quelle fonction...
```

On peut aussi faire apparaître un quadrillage, ou à la fois un quadrillage et les axes des abscisses et des ordonnées par exemple. *Ce sera à vous de voir ce qui permettra d'interpréter le mieux la figure tracée, ce ne sera pas toujours le même choix.*

```
x = np.linspace(0, 2*np.pi, 100) # le␣
 ↪vecteur des abscisses

plt.figure(1) #quand on veut faire␣
 ↪plusieurs figures à la suite dans la même␣
 ↪cellule, on les numérote

plt.plot(x, np.sin(x), label='sinus')
plt.plot(x, np.cos(x), label='cosinus')
plt.grid() # pour avoir un quadrillage
plt.xlabel('x') # pour donner un nom à␣
 ↪l'axe des abscisses
plt.ylabel('y') # pour donner un nom à␣
 ↪l'axe des ordonnées
plt.title('joli graphe') # pour donner un␣
 ↪nom à la figure
plt.legend() # pour indiquer une légende et␣
 ↪donc savoir quelle courbe correspond à␣
 ↪quelle fonction...

plt.figure(2)
# ces quelques lignes, facultatives,␣
 ↪permettent de faire apparaître les axes␣
 ↪habituels des abscisses et des ordonnées
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
# fin des lignes facultatives

plt.plot(x, np.sin(x), label='sinus')
plt.plot(x, np.cos(x), label='cosinus')
plt.grid() # pour avoir un quadrillage
plt.xlabel('x') # pour donner un nom à␣
 ↪l'axe des abscisses
plt.ylabel('y') # pour donner un nom à␣
 ↪l'axe des ordonnées
plt.title('joli graphe') # pour donner un␣
 ↪nom à la figure
plt.legend() # pour indiquer une légende et␣
 ↪donc savoir quelle courbe correspond à␣
 ↪quelle fonction...
```

Exercice 2 - Réprésentations de méthodes d'intégration numérique

Le but de cet exercice est de représenter graphiquement les méthodes des rectangles et la méthode des trapèzes.

Tester, puis analyser le code suivant

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

def rep_int(f,a,b,n,methode):

    # ces quelques lignes, facultatives,␣
↪permettent de faire apparaître les axes␣
↪habituels des abscisses et des ordonnées
    ax = plt.gca()
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.xaxis.set_ticks_position('bottom')
    ax.spines['bottom'].
↪set_position(('data',0))
    ax.yaxis.set_ticks_position('left')
    ax.spines['left'].
↪set_position(('data',0))
    # fin des lignes facultatives

    x = np.linspace(a,b, 100) #le vecteur␣
↪des abscisses
    plt.plot(x,f(x),color='red') #le tracé␣
↪du graphe en rouge de la fonction f

    x,h = np.linspace(a,b, n+1,␣
↪retstep=True) #la subdivision régilière␣
↪du segment [a,b] en n segments.
    if methode=='rectgauche':
        y=f(x)
        titre='à gauche'
    elif methode=='rectdroite':
        y=f(x+h)
        titre='à droite'
        plt.fill_between(x,y,0,step='post',␣
↪facecolor='green',alpha=0.5)␣
↪#représentation des aires des rectangles

    #tracés des cotés des rectangles
    plt.vlines(x[:-1],0,y[:-1])
    plt.vlines(x[1:],0,y[:-1])
    plt.hlines(y[:-1],x[:-1],x[1:])


    plt.title('méthode des rectangles␣
↪'+titre)
    plt.legend(['f','aire'])
```

```
rep_int(np.cos,0,3*np.pi/2,5,'rectgauche')
```

```
rep_int(np.cos,0,3*np.pi/2,5,'rectdroite')
```

Compléter la fonction **rep_int** pour représenter la méthode des rectangles au milieu

```
#
#
#
#
#
#
#
#
#
#
```

Compléter la fonction **rep_int** pour représenter la méthode des trapèzes.

```
#
#
#
#
#
#
#
#
#
#
```

Vous pouvez maintenant utiliser l'onglet **Fichier** –> **Close and Shut Down Notebook** puis dans l'onglet **Nbgrader** –> **Assigments** –> **Download Assigments** cliquer sur le bouton **Submit**