
TP N° 6 : (TP Noté) Constante d'Euler, vélos et animation.

Pour ce travail vous devez déposer un **unique** fichier au format `.ipynb`, dont le nom est `tp_note_hmla310_group_?_prenom_nom.ipynb`, le tout en minuscule. Vous remplirez votre nom, prénom et le groupe qui vous concerne (?=A,B ou C) de manière adéquate. (1 point de malus si le nom du fichier ne suit pas cette forme)

Vous devez charger votre fichier sur Moodle, avant le vendredi 23/10/2020, 23h59.

La note totale est sur **20** points répartis comme suit :

- qualité des réponses aux questions : **14** pts,
- qualité de rédaction et d'orthographe : **1** pts,
- qualité des graphiques (légendes, couleurs) : 1 pt
- style PEP8 valide : **2** pts,
- qualité d'écriture du code (noms de variable clairs, commentaires, code synthétique, etc.) : **1** pt
- Notebook reproductible (i.e., "Restart & Run all" marche correctement sur la machine du correcteur) et absence de bug : **1** pt

Les personnes qui n'auront pas soumis leur devoir sur Moodle avant la limite obtiendront **zéro**.

Rappel : aucun travail par mail ne sera accepté !

Exercice 1. (constante d'Euler)

La constante d'Euler (où bien encore d'Euler-Mascheroni) est une constante, usuellement noté γ, qui est définie comme la limite de la différence entre la série harmonique et le logarithme népérien :

$$\gamma = \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k} - \ln(n)\right) = \lim_{n\to\infty}\left(1+\frac{1}{2}+\cdots+\frac{1}{n} - \ln(n)\right) \approx 0,5772156649. \tag{1}$$

Dans cet exercice, nous nous proposons d'approximer de plusieurs façon cette constante.

1.1. A l'aide d'une boucle `for` et de la fonction `log` de la librairie `numpy`, proposez deux fonctions permettant de calculer une approximation de la constante d'Euler en fonction de n :

   (a) la première fonction, qu'on appellera `approx_constante_euler_croissante` calculera la série harmonique de façon croissante : $1+\frac{1}{2}+\cdots+\frac{1}{n}$.

   (b) la deuxième fonction, qu'on appellera `approx_constante_euler_decroissante` calculera la série harmonique de façon décroissante : $\frac{1}{n}+\cdots+\frac{1}{2}+1$.

   Appliquez vos deux fonctions pour $n = 10^7$. Que constatez vous ? Comment expliquez vous cette différence alors que les opérations effectuées sont identiques ?

1.2. Proposez cette fois une fonction approximant la constante d'Euler en fonction de n, sans boucle for à l'aide de `numpy`. On appelera cette fonction : `approx_constante_euler_numpy`. Cette fonction renverra pour un nombre $n > 0$ donné, un numpy array $[x_1, \ldots, x_n]$ de taille n, dont le terme général vaut $x_i = \left(\sum_{k=1}^{i}\frac{1}{k}\right) - \log(i)$.

La constante d'Euler peut-être exprimée de plusieurs autres façons, dont certaines faisant intervenir des intégrales. En particulier, on s'intéressera à la représentation suivante (admise) :

$$\gamma = \int_0^{\infty}\frac{1}{x}\left(\frac{1}{x+1} - e^{-x}\right)dx \ . \tag{2}$$

Définissons la fonction f sur $]0, +\infty[$

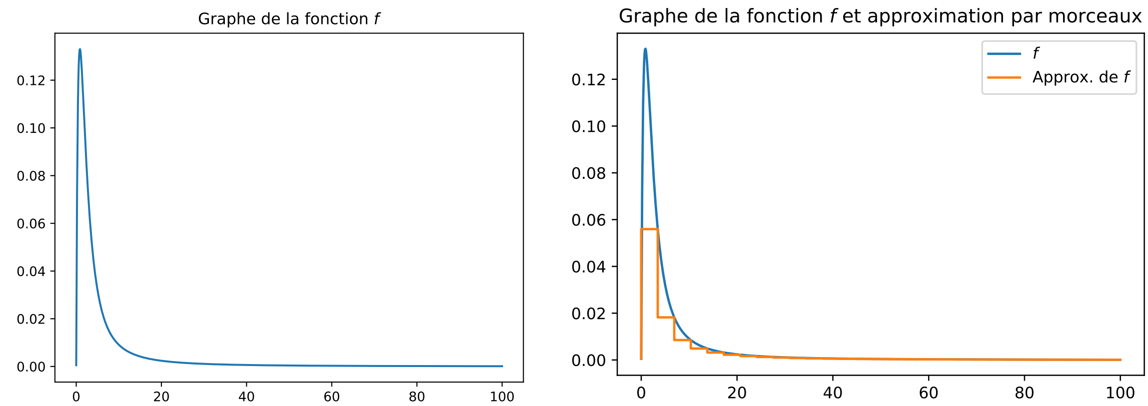
$$f(x) = \frac{1}{x}\left(\frac{1}{x+1} - e^{-x}\right) \quad . \tag{3}$$

Dans toute la suite de l'exercice, on se proposera d'approximer la constante d'Euler à partir de l'intégrale précédente.

1.3. Créez une fonction permettant de calculer $f(x)$ pour tout $x > 0$. On notera cette fonction `fct_a_integre`.
Note : pour l'exponentiel, vous pouvez utilisez `np.exp` où `np` représente le préfixe de `numpy`.

1.4. En utilisant le module `matplotlib`, afficher le graphe de cette fonction entre 0.001 et 100 (avec 1000 points de discrétisation). Vous devez obtenir un graphique similaire à la Figure 1a.

1.5. Afficher (par exemple avec la fonction `step` de `matplotlib`) une approximation constante par morceaux de la fonction précédente entre 0.001 et 100 (avec 30 points de discrétisation), tout en gardant la figure précédente sur le même graphique, comme en Figure 1.



(a) Résultat question 4

(b) Résultat question 6

FIGURE 1 – Graphe de f et de son approximation par une fonction constante par morceaux entre 0.001 et 100.

On rappelle ici la définition des sommes de Riemann. Soit $f : [a, b] \to \mathbb{R}$ une fonction définie en tout point du segment $[a, b]$. On se donne alors $n + 1$ points définis par $x_k = a + k \cdot \frac{b-a}{n}$, avec $0 \leqslant k \leqslant n$. La somme de Riemann de f (d'ordre n) sur $[a, b]$ est alors définie par :

$$S_n(f, a, b) = \sum_{i=1}^{n} (x_i - x_{i-1}) f(x_i) \quad . \tag{4}$$

1.6. Écrire une fonction Python appelée `sum_riemann` qui prend en entrée : f, a, b et n et qui renvoie, en utilisant la fonction `diff`, la somme de Riemann décrite par l'équation (4). Donner alors l'approximation de γ obtenue en appliquant cette fonction avec `fct_a_integre`, $a = 0.001$, $b = 1000$, $n = 1000$

1.7. Créer une nouvelle fonction `sum_riemann_bis` ayant en plus un argument optionnel booléen, appelé `pt_median`, et qui implémente la méthode du point médian[1] si l'argument est `True` et fonctionne comme `sum_riemann` si `pt_median=False`.

1.8. À l'aide de la fonction `interact` du module `ipywidgets`, créer un widget comme dans le TP4 permettant d'afficher une approximation de la fonction f en fonction ayant trois "*sliders*" : 1) la borne minimale de l'intégration (a), 2) la borne maximale de l'intégration (b) et 3) le nombre de points de discrétisation (`n_discr`). On pourra s'inspirer des instructions de la question 5 et du squelette suivant :

1. voir la page : https://fr.wikipedia.org/wiki/Somme_de_Riemann

```
    def plot_approx(a=0.001, b=1000, n_discr=100):
        x = np.linspace(XXX, XXX, XXX endpoint=True) # utilisez question 4
        x_gross = np.linspace(XXX, XXX, XXX, endpoint=True) # utilisez question 6
        plt.plot(XXX, XXX, label='$f$')
        plt.step(XXX, XXX, label='Approx. de $f$')
```

Enfin on affichera dans le titre la valeur de l'approximation obtenue par γ pour les paramètres choisit en prenant `sum_riemann_bis(fct_a_integre, a, b, n_discr, True)`, de sorte que la valeur sera recalculée quand les curseurs varient.

- Données du Totem Albert 1er -

Commençons par charger les données `Donnees_Comptages_Velos_Totem_Albert_1er_verbose.csv`. Ces données donne le nombre de vélo qui sont passés sur la place Albert 1er (Montpellier) au cours d'une journée et ce depuis le début de l'année 2020.

Si besoin, vous devrez peut-être installer le package `download` avec la commande suivante :

```
pip install download # à exécuter dans une cellule si besoin.
```

Sinon, vous pourrez aussi charger les données à la main, et passer la partie téléchargement automatique de la cellule suivante.

Téléchargement automatisé des données :

```
# Download
import os
import numpy as np
import pandas as pd
from download import download
url = "http://josephsalmon.eu/enseignement/datasets/"
name = "Donnees_Comptages_Velos_Totem_Albert_1er_verbose.csv"
path_target = os.path.join(os.getcwd(), name)
download(url + name, path_target, replace=False)
```

Ceci étant fait, les données sont disponibles dans votre répertoire (le vérifier avec `ls` par exemple).

On va maintenant importer ces données avec `pandas` en utilisant la commande suivante :

```
bikes_df = pd.read_csv('Donnees_Comptages_Velos_Totem_Albert_1er_verbose.csv',
                       comment='#', header=0,
                       usecols=["Date", "Heure / Time",
                                "Vélos depuis la mise en service / Grand total",
                                "Vélos ce jour / Today's total"],
                       thousands=' ')
columns_name = bikes_df.columns
bikes_df.rename(columns={"Vélos depuis la mise en service / Grand total": "Cumul",
                         "Vélos ce jour / Today's total": "Jour"}, inplace=True)
bikes_df.head(25)
```

2.1. Remarquez que les colonnes `Date` et `Heure / Time` sont disjointes et sous format textuel. Afin de faire des manipulations simples des jours, mois, années, etc. on se propose d'importer ces données au format de type `datetime` de `pandas`. Effectuez les diverses étapes qui suivent et décrivez l'impact de chaque ligne :

```
time_improved = pd.to_datetime(bikes_df['Date'] +
                               ' ' + bikes_df['Heure / Time'],
                               format='%d/%m/%Y %H:%M:%S')
bikes_df['DateTime'] = time_improved
bikes_df.drop('Date', 1, inplace=True)
bikes_df.drop('Heure / Time', 1, inplace=True)
```

2.2. Que signifie NaT dans la première ligne du tableau `bikes_df` ? Exécuter la commande suivante pour enlever cette ligne atypique, et indexer les lignes par la colonne DateTime :

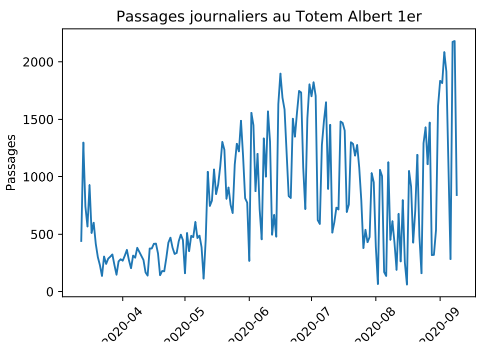
```
bikes_df.dropna(inplace=True)
```

2.3. Investiguez le fichier .csv récupéré. En particulier que se passe-t-il ligne 7 ? (la ligne qui commence par “13/03/2020,10 :02 :00,”). Utiliser et décrire précisément ce que font les instructions suivantes :

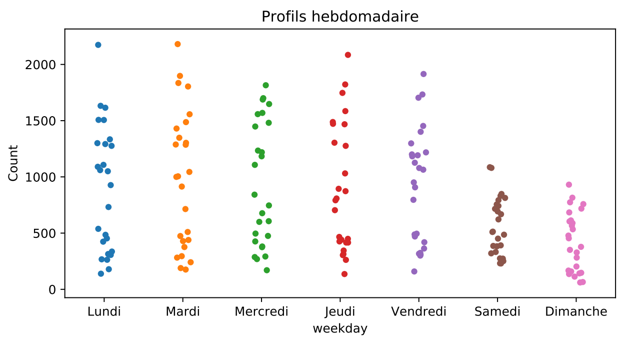
```
bikes_df['Cumul'] = bikes_df['Cumul'].str.replace('\s+','').astype(int)
bikes_df['Jour'] = bikes_df['Jour'].str.replace('\s+','').astype(int)
bikes_df = bikes_df.set_index(['DateTime'])
```

2.4. En utilisant la commande `resample`, afficher l'évolution du nombre de passages au compteur par jour sur toute la durée d'étude (ce qui devrait fournir un graphe comme sur la Figure 2a).

2.5. Afficher un graphique de dispersion par jour de la semaine des passages journaliers au totem (c'est-à-dire illustrer le profils hebdomadaire des passages sur la période d'étude). On pourra de nouveau utiliser la commande `resample` pour afficher la moyenne journalière pour les 7 jours de la semaine.



(a) Évolution sur l'année



(b) Profils hebdomadaire

Figure 2 – Analyse des données du Totem Albert 1er

2.6. Trouver les 3 jours ayant les comptes de passage maximum (respectivement minimum). On pourra utiliser la commande `idxmax` (respectivement `idxmin`) ou encore `argpartition`. Donner votre analyse de ces jours extrêmes.

- Vidéo -

3.1. Écrire un code Python qui permette de créer une vidéo similaire à celle-ci : http://josephsalmon.eu/enseignement/datasets/test.gif. On pourra pour cela s'inspirer d'exemples tel que celui-ci https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/ ou encore consulter l'aide de `matpotlib` : https://matplotlib.org/3.1.1/api/animation_api.html.