TP 13 : séries - déterminant

Exercice 1 (Ecricome 2018)**.**

Pour tout entier naturel n non nul, on pose $u_n = \dfrac{1}{n}$ et $S_n = \sum_{k=1}^{n} u_k$ sa n-ième somme partielle.

On a vu que S est une série divergente. On pose maintenant pour tout $n \in \mathbb{N}^*$, $v_n = S_n - \ln(n)$.

v est donc une soustraction de deux suites qui tendent toutes deux vers $+\infty$: on obtient une F.I.

1. Écrire un script qui demande un entier n à l'utilisateur puis qui calcule et affiche les n premières valeurs de la suite $(u_n)_{n\geqslant 1}$.
2. Compléter votre programme pour qu'il calcule et affiche les valeurs successives de $(S_n)_{n\geqslant 1}$ à la place de celles de $(u_n)_{n\geqslant 1}$.
3. À l'aide des questions précédentes, écrire une fonction Python nommée `v` qui prend en argument un entier naturel n non nul et qui renvoie la valeur de v_n.
4. Affichez à l'aide de `plot` les 500 premières valeurs de $(v_n)_{n\in\mathbb{N}}$.

   L'étude mathématique permet de confirmer que $(v_n)_{n\in\mathbb{N}^*}$ est décroissante et d'affirmer qu'elle est minorée par $1-\ln(2)$. D'après le théorème de convergence monotone, elle est donc convergente. On note γ sa limite. On démontre enfin que

$$\forall n \in \mathbb{N}^*\ ,\ |v_n - \gamma| \leqslant \frac{1}{n}\ .$$

5. On rappelle que l'instruction `floor(x)` renvoie la partie entière d'un réel x et on suppose que la fonction `v` de la question 3) a été correctement programmée. Expliquer l'intérêt et le fonctionnement du script ci-dessous :

```
eps=eval(input('Entrer un reel strictement positif : '))
n=floor(1/eps)+1
print(v(n))
```

6. Compléter votre script afin d'obtenir une approximation à 10^{-4} près de γ.

Exercice 2 (HEC 2019)**.**

La fonction Python suivante permet de multiplier la i-ème ligne L_i d'une matrice A par un réel sans modifier ses autres lignes, c'est-à-dire de lui appliquer l'opération élémentaire $L_i \leftarrow aL_i$ (où $a \neq 0$).

```
def multlig(a,i,A):
    (p,n) = A.shape
    B = A
    for j in range(n):
        B[i,j] = a*B[i,j]
    return B
```

1. Donner le code Python de deux fonctions `addlig` (d'arguments b, i, j, A) et *echlig* (d'arguments i,j,A) permettant d'effectuer respectivement les deux autres opérations élémentaires sur les lignes d'une matrice : $L_i \leftarrow L_i + bL_j$ $(i \neq j)$ et $L_i \leftrightarrow L_j$ $(i \neq j)$.
2. Expliquer pourquoi la fonction `multligmat` suivante retourne le même résultat B que la fonction `multlig`.

```python
def multligmat(a,i,A):
    (p,n) = A.shape
    D = np.eye(p)
    D[i,i] = a
    return np.dot(D,A)
```

3. Trouver des façons matricielles d'effectuer les deux autres opérations élémentaires sur les lignes d'une matrice, et les coder en Python sous les noms `echligmat` et `addligmat`.
4. Les matrices utilisées pour réaliser les opérations sur les lignes sont elles inversibles ?

La commande ***rd.random([p,n])*** *génère une matrice de taille $p \times n$ dont tous les coefficients sont des réels aléatoirement choisis entre 0 et 1. On peut démontrer (c'est un résultat difficile néanmoins) que la probabilité qu'une matrice aléatoire comme celle-ci a pour probabilité 1 d'être inversible.*

Exercice 3 (Déterminants de matrices et inversibilité)**.**

1. Générer une matrice aléatoire A de taille 100×100 et faites calculer l'inverse de cette matrice par Python.

   Remarquez que le résultat est obtenu très rapidement malgré la taille de la matrice.
   On va coder nous-même une méthode d'inversion de matrices généralisant la formule d'inversion des matrices 2×2. Nous comparerons finalement notre méthode avec celle utilisée par Python. On va exploiter le résultat suivant (difficile donc admis).
   Définition :

   À toute matrice $A = (a_{i,j})_{1 \leqslant i,j \leqslant n} \in \mathcal{M}_n(\mathbb{R})$ on associe un unique réel $\det(A)$ défini de la manière suivante :
   • Si $n = 1$, on pose simplement $\det(A) = a_{1,1}$.
   • Si $n > 1$, alors pour tout i, j de $[\![1; n]\!]$ on note A_{ij} la matrice A dans laquelle on a supprimé la ligne i et la colonne j. On pose alors

$$\det(A) = a_{1,1}\det(A_{11}) - a_{1,2}\det(A_{12}) + \cdots + (-1)^{n-1}\det(A_{1n}) = \sum_{k=1}^{n} a_{1,k}(-1)^{k-1}\det(A_{1k})$$

   Autrement dit le déterminant est défini par récurrence : pour calculer un déterminant d'une matrice de taille n, on calcule n déterminants de matrices de taille $n-1$, etc... jusqu'à devoir calculer seulement des déterminants de matrices de taille 1.
2. Vérifiez que cette formule, dans le cas $n = 2$, re-donne bien le déterminant d'une matrice 2×2 utilisé pour l'inversion des matrices carrées de taille 2.
3. *Pour calculer, à partir d'une matrice A, la matrice A_{ij}, nous allons exploiter la souplesse de manipulation des matrices en Python.*
4. Recopiez le code suivant dans un script Python :

```python
def deter(A):
    (p,n)=A.shape
    if n==1 :
        return A[0,0]
    else :
        somme = 0
        B = A[1:n,:]
        for k in range (n):
            indices = [i for i in range(n)]
            indices.remove(k)
            somme += (-1)**k*A[0,k]*deter(B[:,indices])
    return somme
```

5. Testez cette fonction sur la matrice $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$, puis sur la matrice aléatoire A de l'exercice 1.

   La fonction correspondante en Python est la fonction **np.linalg.det**. Comparez **np.linalg.det** et **deter**.

   Une matrice est inversible si et seulement si son déterminant est non nul, tout comme dans le cas 2×2. *Nous avons aussi une formule pour l'inverse de la matrice.*

   Proposition :

   Soit A de déterminant non nul, on note A_{ij} à nouveau les sous-matrices de A dans les quelles on a supprimé la i-ième ligne et la j-ième colonne. Alors l'élément i, j de la transposée de A^{-1} est $\dfrac{(-1)^{i+j}\det(A_{ij})}{\det(A)}$.

6. Recopiez le code suivant dans un script Python :

```
def inverse(A):
    (p,n) = A.shape
    B = np.zeros([p,n])
    for i in range (p):
        for j in range(n):
            lignes = [k for k in range(p)]
            lignes.remove(i)
            colonnes = [k for k in range(n)]
            colonnes.remove(j)
            C = A[lignes,:]
            C = C[:,colonnes]
            B[i,j] = (-1)**(i+j)*deter(C)
    return B.transpose()/deter(A)
```

7. Testez cette fonction sur la matrice $\begin{pmatrix} 1 & 2 \\ 3 & 7 \end{pmatrix}$ dont vous calculerez l'inverse sur feuille pour vérifier.
8. Testez maintenant sur la matrice A aléatoire de l'exercice 1 et comparez avec la fonction Python **np.linalg.inv**. Que remarquez-vous ?

Formulaire :

Vous aurez besoin des commandes suivantes (dans le désordre)

```
from math import *
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rd
rd.random(), rd.random(n), rd.random([p,n])
np.linspace
np.linalg.inv(A)
for ... in ... :
def ... ( ... ) :
return
print( ... )
if ... :
else :
log
[ ... for ... in ...]
plt.plot( ... , ... )
plt.show()
```