DS 5 - Math/Info

Exercice 1. Montrer que la matrice $P = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$ est inversible et calculer son inverse.

Exercice 2. Soient $A = \begin{pmatrix} -1 & 5 & -3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ et $P = \begin{pmatrix} 9 & 1 & -1 \\ -3 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$

1. Résoudre, pour tout $\lambda \in \mathbb{R}$, le système : $(A - \lambda I_3)X = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ avec $X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$.
2. Soit $Q = \begin{pmatrix} 1 & -2 & 1 \\ 3 & 10 & 3 \\ -4 & -8 & 12 \end{pmatrix}$. Calculer PQ.
3. En déduire que P est inversible et donner P^{-1} en fonction de Q.
4. Calculer $P^{-1}AP$.
5. En déduire que $A^n = PT^nP^{-1}$ pour tout $n \in \mathbb{N}$ où T est une matrice triangulaire à déterminer.
6. Montrer que pour tout $n \in \mathbb{N}$,
$$T^n = \begin{pmatrix} (-3)^n & 0 & 0 \\ 0 & 1 & -n \\ 0 & 0 & 1 \end{pmatrix}$$
7. On considère une suite $(u_n)_{n\in\mathbb{N}}$ définies par : $\begin{cases} u_0 = 0,\ u_1 = 0,\ u_2 = 1 \\ \forall n \in \mathbb{N},\ u_{n+3} = -u_{n+2} + 5u_{n+1} - 3u_n. \end{cases}$
   (a) On pose : $X_n = \begin{pmatrix} u_{n+2} \\ u_{n+1} \\ u_n \end{pmatrix}$. Donner la relation qui lie X_{n+1}, X_n et A pour tout $n \in \mathbb{N}$.
   (b) En déduire que pour tout $n \in \mathbb{N}$: $X_n = A^nX_0$.
   (c) En déduire l'expression explicite de la suite $(u_n)_{n\in\mathbb{N}}$.

Exercice 3. On se place dans l'espace $\mathbb{R}^3$ et on considère le plan $\mathcal{P}$ d'équation cartisenne :

$$\mathcal{P} : x + 2y - z = 1$$

Pour tout ce problème, on fixe un point $A = (\alpha, \beta, \gamma)$ et on note $H(\lambda, \mu, \nu)$ son projeté orthogonal sur $\mathcal{P}$. Le but de ce problème est de déterminer (λ, μ, ν) en fonction de (α, β, γ)

1. Donner un vecteur normal à $\mathcal{P}$.
2. Déterminer une équation paramétrique de la droite $\mathcal{D}$ passant par A et orthogonale à $\mathcal{P}$.
3. Montrer que le plan $\mathcal{P}_2$ d'équation $\mathcal{P}_2 : -2x + y = \beta - 2\alpha$ contient la droite $\mathcal{D}$.
   On admet que l'on peut montrer de la même manière que le plan $\mathcal{P}_3$ d'équation $\mathcal{P}_3 : x + z = \alpha + \gamma$ contient la droite $\mathcal{D}$.
4. Justifier que $\mathcal{D} = \mathcal{P}_2 \cap \mathcal{P}_3$.
5. En déduire que les coordonnées de H vérifient un système linéaire qu'on peut écrire sous forme matricielle sous la forme :

   $$M_1 \begin{pmatrix} \lambda \\ \mu \\ \nu \end{pmatrix} = M_2 \quad \text{où } M_1 = \begin{pmatrix} 1 & 2 & -1 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \text{ et } M_2 \in M_{3,1}(\mathbb{R}) \text{ est une matrice colonne à déterminer.}$$

   M_2 s'exprimera en fonction de α, β, γ.
   On admet que M_1 est inversible et que

   $$M_1^{-1} = \frac{1}{6} \begin{pmatrix} 1 & -2 & 1 \\ 2 & 2 & 2 \\ -1 & 2 & 5 \end{pmatrix}$$

6. En déduire que

   $$\begin{pmatrix} \lambda \\ \mu \\ \nu \end{pmatrix} = P_1 \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} + P_2 \text{ où } P_1 = \frac{1}{6} \begin{pmatrix} 5 & -2 & 1 \\ -2 & 2 & 2 \\ 1 & 2 & 5 \end{pmatrix} \text{ et } P_2 = \frac{1}{6} \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$$

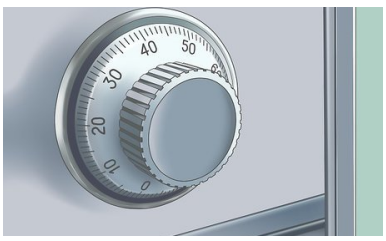
   On a ainsi répondu à la question principale. Deux questions supplémentaires vous sont proposées pour voir l'interaction entre la géométrie et les matrices.
7. Calculer P_1^2 et $P_1 P_2$
8. En déduire que

   $$P_1 \begin{pmatrix} \lambda \\ \mu \\ \nu \end{pmatrix} + P_2 = \begin{pmatrix} \lambda \\ \mu \\ \nu \end{pmatrix}$$

   et en expliquer la signification géométrique.

Exercice 4.

Olivier voudrait voler le contenu du coffre fort de Yann, mais il ne connaît pas la combinaison qu'il a choisie. Il s'est cependant procuré le manuel du coffre, qui indique que la combinaison est nécessairement une suite de 3 à 8 entiers, tous compris entre 0 et 99, à rentrer dans l'ordre.



On note k la taille de la combinaison du coffre : on a donc $k \in [\![3, 8]\!]$.

Les questions sont indépendantes. À chaque fois, on demande de justifier (briévement) mais pas de faire les applications numériques, sauf mention explicite du contraire.

1. Combien de combinaisons sont possibles pour le coffre fort si on sait qu'elle est constituée de 4 nombres ?

2. Combien de combinaisons sont possibles en tout si on ne connaît pas k ?
3. Combien de combinaisons sont possibles si on sait que $k = 3$ et que les 3 nombres sont tous différents ?
4. Combien de combinaisons sont possibles si on sait que $k = 4$ et que la suite de nombres est strictement croissante ?
5. Olivier sait que Yann a choisi une combinaison à 5 nombres ($k = 5$), et il connaît aussi ses nombres préférés.
   (a) Combien de combinaisons possibles contiennent au moins une fois le nombre 77 ?
   (b) Combien de combinaisons contiennent exactement une fois chacun des nombres 1, 7, 17, 77 et 99 ? On donnera la valeur numérique.
6. **Python** On représente une combinaison en Python par une liste d'entiers.
   (a) Écrire une fonction `verifie(L)` qui prend en entrée une liste d'entiers `L`, et qui renvoie `True` si la liste correspond à une combinaison valide selon le manuel du coffre, et `False` sinon.
   (b) Écrire une fonction `strict_croissante(Comb)` qui prend en argument une liste `Comb` représentant une combinaison, et qui vérifie si elle est strictement croissante.

Python

Exercice 5. On rappelle qu'une anagramme d'un mot est un mot obtenu en réarrangeant les lettres d'un autre mot. Par exemple :

`IMAGINER` et `MIGRAINE`

On ne supposera pas que les mots sont dans le dictionnaire français, ainsi

`IMAGINER` et `GRAINEMI`

seront aussi deux anagrammes valides dans ce problème.

Dans cette exercice, les chaînes de caractères ne contiendront que des lettres majuscules et pas de caractères spéciaux ou d'espaces.

Les fonctions `count` et `index` sont interdites. Les fonctions précédemment codées dans la copie, et même celles des questions admises, peuvent être utilisées dans les questions suivantes.

```
1 def anagramme_faux(s,t):
2     if len(s)!=len(t):
3         return(False)
4     for lettre in s:
5         if lettre not in t:
6             return(False)
7     return(True)
```

Trouver deux chaînes de caractères `ch1` et `ch2` qui ne sont pas anagrammes l'une de l'autre et qui pourtant vérifient `anagramme_faux(ch1,ch2)`—>`True` (je vous conseille de faire la question 2 avant de réfléchir à celle-ci pour comprendre le problème)

2. Détection si deux chaînes sont des anagrammes :
   (a) Écrire une fonction `NbAppar(ch,a)` qui prend en argument une chaine de caractères et un caractère et qui renvoie le nombre de fois que le caractère `a` apparait dans `ch`.

   Par exemple, `NbAppar('ANAGRAMME', 'A')` renvoie 3.
   (b) Soient `ch1` et `ch2` deux chaînes de caractères de même longueur. Donner une condition nécessaire et suffisante portant sur le nombre d'apparitions de chaque lettre pour que ces deux chaînes de caractères soient des anagrammes.
   (c) On considère la fonction :

```
def mystere(s):
    L=[]
    for l in s:
        if l not in L:
            L.append(l)
    return L
```

Que retourne `mystere('OLIVIER')` ?

(d) Écrire une fonction `VerfiAnag(ch1,ch2)` qui prend en argument deux chaînes de caractères et qui renvoie `True` si `ch1` et `ch2` sont des anagrammes et `False` sinon.

3. Dénombrement informatique des anagrammes d'un mot. Comme pour le dénombrement mathématiques, nous allons considérer *a priori* les lettres identiques comme différentes, puis nous allons enlever les doublons.

(a) Parmi les quatre fonctions suivantes, déterminer l'unique fonction qui prend en argument une chaîne de caractère `ch` et un caractère `a` et qui renvoie la liste chaînes de caractères obtenue en insérant à chaque position possible de `ch`.

Par exemple, pour `ch='BO'` et `a='A'`, on doit obtenir `['ABO', 'BAO', 'BOA']`. Pour `ch='BO'` et `a='B'`, on doit obtenir `['BBO', 'BBO', 'BOB']` (le premier et le deuxième 'BBO' correspondent respectivement à l'ajout du 'B' à la position 0 puis à la position 1)

Pour celles qui ne réalisent pas ce qui est demandé on donnera un contre exemple d'une chaîne `ch` et d'un caractère `a` et on donnera le résultat de la fonction.

```
def Inser1(ch, a) :
    L = []
    for k in range(len(ch)) :
        L.append(ch[:k]+a+ch[k:])
    return L
```

```
def Inser2(ch,a) :
    L = []
    for k in range(len(ch)+1) :
        L.append(ch[:k]+a+ch[k:])
    return L
```

```
def Inser3(ch,a) :
    L = []
    for k in range(len(ch)) :
        L.append(ch[:k]+a+ch[k+1:])
    return L
```

```
def Inser4(ch,a) :
    L = []
    for k in range(len(ch)+1) :
        L.append(ch[:k]+a+ch[k+1:])
    return L
```

(b) Utiliser la fonction d'insertion précédente pour écrire une fonction `InserListe(L,a)` qui prend en argument une liste de chaines de caractères et qui renvoie la liste des chaines de caractères obtenue en insérant à chaque position possible de chacune des chaines de caractères de `L`.

Par exemple, `InserListe(['OB', 'BO'], 'B')` renvoie `['BOB','OBB','OBB','BBO','BBO','BOB']`.

(c) Compléter la fonction `ListeAnag(ch)` qui prend en argument une chaîne de caractères et afin qu'elle renvoie la liste des anagrammes de la chaîne de caractères (éventuellement avec répétition).

Par exemple, `ListeAnag('BOB')` devra renvoyer `['BOB','OBB','OBB','BBO','BBO','BOB']`.

```
def ListeAnag(ch):
    L=['']
    n=...
    for i in range(n):
        L=inserListe(... , ...)
    return
```

(d) Ecrire une fonction `SansRepet(L)` qui prend en argument une liste `L` pour et qui renvoie la même liste sans répétition.

Par exemple, `SansRepet(['BOB','OBB','OBB','BBO','BBO','BOB']` renvoie `['BOB','OBB','BBO']`.

(e) Utiliser les fonctions précédentes pour écrire une fonction `NbAnag(ch)` qui prend en argument une chaîne de caractères et qui renvoie le nombre d'anagrammes de ce mot.

(f) Quelle valeur va retourner `NbAnag('OLIVIER')` ? (à exprimer à l'aide de factorielle - ce n'est pas une question d'info)