Feuille 5 : Tableaux et matrices.

Nous présentons dans ce TP les bibliothèques `numpy` et `numpy.linalg`.

1 Rappels sur les vecteurs

Commençons par importer la bibliothèque `numpy` à l'aide de la commande :

```
>>> import numpy as np
```

Une fois cette commande exécutée, nous avons maintenant accès aux fonctions usuelles mathématiques.

Définition 1 *Les fonctions usuelles mathématiques accessibles par la bibliothèque* ***numpy*** *sont (entre autres) :*

np.abs *(valeur absolue),* ***np.cos*** *(cosinus),* ***np.sin*** *(sinus),* ***np.log*** *(logarithme népérien)*
np.exp *(exponentielle),* ***np.sqrt*** *(racine carrée),* ***np.floor*** *(partie entière).*

Mise en garde.

Noter la syntaxe `np.log` (et non `np.ln`) pour la fonction logarithme népérien.

Exemple. Éxécuter les instructions suivantes dans la console.

```
>>> np.pi ; np.e
>>> np.floor(np.sqrt(5))
>>> np.sin(1.0); np.tan(np.pi/4)
```

1.1 Les vecteurs

Définition 2 *Pour définir le vecteur-ligne* $(u_1, \dots, u_n)$ *(de type* ***array****, tableau en anglais), on utilise la commande :*

np.array([u1,...,un])

Exemple. Éxécuter l'instruction suivante dans la console :

```
>>> u = np.array([1,5,-2]) ; u
```

Remarque. Un vecteur (et plus généralement un tableau) ne peut contenir que des éléments de même type. Si on essaie de fabriquer un tableau à partir de types différents, `Python` essaie de convertir toutes les données vers un type commun et retourne une erreur en cas d'échec.

```
>>> np.array([1 , 2, 3.5 , False ])
array([ 1. , 2. , 3.5 , 0. ])
```

Définition 3
- *L'instruction* ***np.arange(a,b,r)*** *renvoie un vecteur ligne dont les coefficients sont en progression arithmétique de raison* `r`*, de premier terme le réel* `a` *et ne dépassant pas le réel* `b`*.*
- *L'instruction* ***np.linspace(a,b,n)*** *renvoie un vecteur ligne dont les* `n` *coefficients sont en progression arithmétique, le premier étant le réel* `a` *et le dernier le réel* `b`*.*
- ***np.ones(n)*** *renvoie le vecteur ligne dont les* `n` *coefficients sont égaux à 1.*
- ***np.zeros(n)*** *renvoie le vecteur ligne dont les* `n` *coefficients sont égaux à 0.*

Remarque. Les instructions `np.arange` et `range` sont *a priori* semblables. Noter cependant ce qui les distinguent :
- l'objet renvoyé : un vecteur pour l'instruction `np.arange` quand `range` ne fait qu'énumérer des entiers sans provoquer d'affichage ;
- les paramètres de `np.arange` ne sont pas nécessairement des entiers.

Noter également que la commande `np.arange(a,b,1)` peut s'écrire plus simplement `np.arange(a,b)`.

Exemples. Éxécuter les instructions suivantes dans la console.

```
>>> np.arange(3,12,2)
>>> np.arange(3,6) ; np.arange(5)
>>> np.linspace(-5,8,6)
>>> np.ones(3)
```

Proposition 1 (Extraction et modification d'éléments d'un vecteur) *Soit* u *un vecteur dont les coefficients sont* $u_0, \dots, u_n$*.*
- *Pour extraire le coefficient* u_i*, on écrit :* ***u[i]****.*
- *Pour modifier le coefficient* u_i *par le réel* a*, on écrit :* ***u[i] = a****.*

Mise en garde.

On notera que l'indexation des coefficients d'un vecteur commence à 0 (comme toujours pour `Python`).

Exemples. Éxécuter les instructions suivantes dans la console.

```
>>> u
>>> u[1]
>>> u[1] = 0 ; u
```

Proposition 2 (Opérations sur les vecteurs coefficient par coefficient) *Soient u et v deux vecteurs dont les coefficients sont respectivement $u_0, \dots, u_n$ et $v_0, \dots, v_n$.*

- *Pour obtenir le vecteur $a \times u$ (a réel), on écrit :* `a*u`.
- *Pour obtenir le vecteur $u + v$, on écrit :* `u+v`.
- *Pour obtenir le vecteur $(u_0 \times v_0, \dots, u_n \times v_n)$, on écrit :* `u*v`.
- *Pour obtenir le vecteur $\left(\frac{u_0}{v_0}, \dots, \frac{u_n}{v_n}\right)$, on écrit :* `u/v`.
- *Pour obtenir le vecteur $(u_0^a, \dots, u_n^a)$ (a réel), on écrit :* `u**a`.
- *Si `f` est une fonction prédéfinie, pour obtenir le vecteur $(f(u_0), \dots, f(u_n))$, on écrit :* `f(u)`.

Exemples. Éxécuter les instructions suivantes dans la console.

```
>>> u = np.arange(3,0,-1); v = np.arange(4,9,2) ; w = 2*np.ones(4)
>>> np.cos(u)
>>> u+v
>>> v/u
>>> v/w
>>> u**(-1)
>>> t = np.arange(1.,8.,2.) ; t**(-1)
```

Remarques.

- Les opérations sur les vecteurs ne sont valides que si les calculs ont du sens. Par exemple, l'instruction `v/w` n'a pas de sens car les vecteurs ne sont pas de même dimension. Elle n'aurait pas non plus de sens si l'un des coefficients du vecteur w était nul.
- Le langage `Python` étant typé, si on entre un vecteur `u` à coefficients entiers (donc de type `int`) et que l'on tape `u**(-1)`, `Python` renvoie un message d'erreur car les résultats ne sont pas de même type. Pour pallier cet inconvénient, on entrera les coefficients suivis d'un point pour que `Python` les interprète en type `float`.
- On peut donc appliquer à un vecteur (et plus généralement à un tableau) la plupart des fonctions fournies par le module `numpy` : la fonction s'applique alors à tous les éléments du tableau en une seule commande (inutile de faire une boucle `for`). De plus, cette exécution est optimisée (elle est beaucoup plus rapide que celle d'une boucle `for` réalisant la même opération).

2 Les matrices

Définition 4 *Pour déclarer la matrice $A = (a_{i,j})_{(i,j)\in[\![1,n]\!]\times[\![1,p]\!]}$ (de type `array`), on écrit :*

`np.array([[`$a_{1,1}$`, ..., `$a_{1,p}$`], ..., [`$a_{n,1}$`, ..., `$a_{n,p}$`]])`

Exemple. Éxécuter les instructions suivantes dans la console :

```
>>> A = np.array([[1,4],[-3,0]]) ; A
>>> X = np.array([[6],[-4]]) ; X
```

Définition 5

- `np.ones((n,p))` *renvoie la matrice de $\mathcal{M}_{n,p}(\mathbb{R})$ dont tous les éléments sont égaux à 1.*
- `np.zeros((n,p))` *crée la matrice de $\mathcal{M}_{n,p}(\mathbb{R})$ dont tous les éléments sont égaux à 0.*
- `np.eye(n,p)` *crée la matrice de $\mathcal{M}_{n,p}(\mathbb{R})$ dont tous les éléments sont nuls sauf ceux dont le numéro de ligne est égal au numéro de colonne (ce sont les éléments diagonaux si $n = p$) qui valent 1.*

Syntaxe.

Avec `np.eye`, il faut des parenthèses simples alors qu'avec `np.ones` et `np.zeros`, il faut des parenthèses doubles.

Proposition 3 (Extraction d'éléments d'une matrice) *On suppose que l'on a déclaré une matrice A.*
- *L'instruction* `a,b=np.shape(A)` *affecte à la variable* `a` *le nombre de lignes et à la variable* `b` *le nombre de colonnes de la matrice A.*
- *Pour extraire le coefficient de la ligne d'indice i et de la colonne d'indice j, on écrit :* `A[i,j]`.
- *Pour extraire la ligne d'indice i, on écrit :* `A[i,:]`.
- *Pour extraire la colonne d'indice j, on écrit :* `A[:,j]`.

Remarque. L'indexation des lignes et des colonnes d'une matrice commence à 0.

Exemples. Éxécuter les instructions suivantes dans la console :

```
>>> np.shape(A) ; np.shape(X)[0] ; np.shape(X)[1]
>>> B = np.array([np.arange(1,4),np.arange(4,7),np.arange(7,10)]); B
>>> B[1,1] ; B[2,:] ; B[:,0] ; B[0:2,1:3]
```

Remarques.
- Le résultat quand on extrait une colonne est une ligne. On peut remédier à cela en remplaçant la commande `B[:,0]` par `B[:,0:1]`.
- Quand un tableau `tab1` est affecté à une variable `tab2`, les tableaux `tab1` et `tab2` partagent les mêmes données. Ils ne sont pas autonomes. Toute modification de l'un se répercute sur l'autre. Même chose lorsqu'on extrait un sous-tableau d'un tableau. Pour éviter cela et disposer d'un tableau autonome, on peut utiliser la commande `tab2 = tab1[:,:].copy()`.

Proposition 4 (Modification d'éléments d'une matrice) *On suppose que l'on a déclaré une matrice A.*
- *Pour remplacer l'élément de A situé à la ligne d'indice i et à la colonne d'indice j par un réel b, on écrit :* `A[i,j] = b`.
- *Pour remplacer la ligne d'indice i de A par L, on écrit :* `A[i,:] = L`.
- *Pour remplacer la colonne d'indice j de A par C, on écrit :* `A[:,j] = C`.

Exemple. Éxécuter les instructions suivantes dans la console :

```
>>> B
>>> B[0,0] = 0 ; B
>>> B[:,1] = np.zeros((1,3)) ; B
```

Proposition 5 (Opérations sur les matrices coefficient par coefficient) *Soient $A = (a_{i,j})$ et $B = (b_{i,j})$ deux matrices* ***de même format*** *déclarées dans les variables* `A` *et* `B`.
- *Pour obtenir la matrice $a \times A$ (a réel), on écrit :* `a*A`.
- *Pour obtenir la matrice $A + B$, on écrit :* `A+B`.
- *Pour obtenir la matrice $C = (a_{i,j} \times b_{i,j})$, on écrit :* `A*B`.
- *Pour obtenir la matrice $C = (a_{i,j}/b_{i,j})$, on écrit :* `A/B`.
- *Pour obtenir la matrice $C = (a_{i,j}^{b_{i,j}})$, on écrit :* `A**B`.
- *Si f est une fonction prédéfinie, pour obtenir la matrice $(f(a_{i,j}))$, on écrit :* `f(A)`.

Remarques.
- Ces opérations sur les matrices ne sont valides que si les calculs ont un sens (même format des matrices, opérations bien définies).
- Pour l'opération `A**B`, il faut faire attention au typage si certains éléments de B sont négatifs, en entrant tous les éléments de A et B sous forme de flottants.

Exemples. Taper les instructions suivantes :

```
>>> A = np.array([[1,4],[0,9]]) ; B = np.array([[2,-1],[3,1]])
>>> A/B
>>> A**(1/2); np.sqrt(A)
>>> C = A+1 ; C
```

Proposition 6 (Opérations matricielles) *Soient A et B deux matrices déclarées dans les variables* `A` *et* `B`.
- *Pour obtenir ${}^tA$ (la matrice transposée de A), on écrit :* `np.transpose(A)`.
- *Pour obtenir la matrice $A \times B$ (produit matriciel), on écrit :* `np.dot(A,B)` *(sous réserve que le produit ait un sens).*

Proposition 7 (Autres opérations sur les matrices) *Soit A une matrice déclarée dans la variable* `A`.
- *Pour obtenir la somme de tous les coefficients de la matrice A, on écrit :* `np.sum(A)`.
  On définit de même les commandes `np.prod(A)`, `np.max(A)`, `np.min(A)`.
- *Pour obtenir la moyenne de tous les coefficients de la matrice A, on écrit :* `np.mean(A)`.
- *Pour obtenir les somme cumulée des coefficients de la matrice A, on écrit :* `np.cumsum(A)`.
  Les sommes cumulées s'effectuent en parcourant la matrice de gauche à droite, de haut en bas, et sont renvoyées sous forme d'un vecteur. On définit de même la commande `np.cumprod(A)`.

Exemples. Taper les instructions suivantes :

```
>>> A ; X ; np.dot(A,X)
>>> C ; np.sum(C) ; np.mean(C) ; np.max(C[1,:])
>>> np.cumsum(C) ; np.cumprod(C)
>>> np.min(C,0) ; np.cumsum(C,0) ; np.cumprod(C,1)
```

Remarque. Les opérations `np.sum`, `np.prod`, `np.max`, `np.min`, `np.mean`, `np.cumsum`, `np.cumprod` peuvent s'appliquer sur une matrice entière, ou bien pour chaque colonne (par exemple `np.prod(C,0)`) ou chaque ligne (par exemple `np.max(C,1)`).

Proposition 8 (Comparaison de matrices) — *Si A et B sont deux matrices de même format, l'instruction `A==B` renvoie une matrice de même format que A (ou B) dont les éléments sont `True` ou `False` selon que les coefficients correspondants de A et B à cette même place sont égaux ou non.*
— *Si tous les éléments de B sont égaux à un même réel x, on peut écrire simplement `A==x`.*
— *On définit de même les matrices booléennes `A>B`, `A>=B`, `A<B`, `A<=B` et `A!=B`.*

Proposition 9 (Opération sur les matrices de booléens) *Si A est une matrice dont les composantes sont des booléens, alors :*
— *la commande `np.sum(A)` renvoie le nombre de booléens qui ont pris la valeur `True`,*
— *la commande `np.mean(A)` renvoie la proportion de booléens qui ont pris la valeur `True`,*

Exemple. Taper les instructions suivantes :

```
>>> A = np.array([[1,4],[0,9]]) ; B = np.array([[2,-1],[3,1]])
>>> A > B ; np.sum(A > B)
>>> A == 0 ; np.mean(A <= 5)
```

3 Le module `numpy.linalg` de calcul d'algèbre linéaire

C'est une sous-bibliothèque du module `numpy`. Elle donne accès à de nouvelles commandes matricielles et permet notamment de résoudre des systèmes linéaires.

Définition 6 *On importe la bibliothèque `numpy.linalg` en écrivant l'une ou l'autre des instructions suivantes (on privilégiera la deuxième) :*

`from numpy.linalg import *` ou `import numpy.linalg as al`

Proposition 10 (Nouvelles commandes matricielles) *On suppose que l'on a déclaré une matrice A.*
— *La commande `al.inv(A)` renvoie l'inverse de A (A est inversible, bien sûr).*
— *La commande `al.matrix_rank(A)` renvoie le rang de A.*
— *Si la matrice A est une matrice carrée et n un entier relatif, la commande `al.matrix_power(A,n)` renvoie A^n (si $n < 0$, il faut bien sûr que A soit inversible).*
— *Si la matrice A est carrée, la commande `al.eig(A)` renvoie les valeurs propres et les vecteurs propres (eigenvalues et eigenvectors en anglais) de la matrice A.*

Proposition 11 (Résolution de systèmes linéaires) *Si A est une matrice carrée inversible et b est une matrice de même taille ou bien un vecteur colonne ayant le même nombre de lignes que A, la commande `al.solve(A,b)` renvoie la solution du système associé à l'équation $Ax = b$.*

Remarque. Si la matrice A n'est pas carrée ou bien si elle n'est pas inversible, `Python` renvoie un message d'erreur, même si l'équation possède une solution.

4 Tableau récapitulatif des fonctions utiles pour les matrices

Je vous fais un tableau récapitulatif des principales fonctions utiles, et de leur application sur les matrices

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{pmatrix} \text{ et } B = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 3 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

(je rappelle que l'on a défini A en Python par la commande `A = np.array([[1,2,3],[0,1,4],[5,6,0]])` et B par la commande `B = np.array([[2,1,1],[1,3,2],[1,1,1]])`).

Fonction	**Effet**	**Exemple sur A**
`A[i,j]`	Extraction d'un élément	`A[2,0]` $\to 5$
`A[i,:]`	Extraction d'une ligne	`A[1,:]` $\to$ `array([0,1,4])`
`A[:,j]`	Extraction d'une colonne	`A[:,2]` $\to$ `array([3,4,0])`
`A[i,:]= np.array([...])`	Modification d'une ligne	`A[1,:]= np.array([1,2,2])` donne $A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 2 \\ 5 & 6 & 0 \end{pmatrix}$
`A[:,j]= np.array([...])`	Modification d'une colonne	`A[:,2]= np.array([[1,2,2]])` donne $A = \begin{pmatrix} 1 & 2 & 1 \\ 1 & 2 & 2 \\ 5 & 6 & 2 \end{pmatrix}$
`a*A`	Multiplication par un scalaire	`3*A` $\to$ `array([[3,6,9],[0,3,12],[15,18,0]])`
`A+B`	Somme matricielle (les matrices doivent avoir la même taille)	`A+B` $\to$ `array([[3,3,4],[1,4,6],[6,7,1]])`
`A*B`	Multiplication coefficient par coefficient	`A*B` $\to$ `array([[2,2,3],[0,3,8],[5,6,0]])`
`A/B`	Division coefficient par coefficient	`A/B` $\to$ `array([[0.5,2.0,3.0], [0.0,0.33333333,2.0],[5.0,6.0,0.0]])`
`np.dot(A,B)` ou `A@B`	Produit matriciel	`np.dot(A,B)` $\to$ `array([[7,14,9],[4,13,10],[12,17,11]])`
`np.shape(A)`	Dimensions de la matrice	`np.shape(A)` $\to (3,3)$
`np.sum(A)`	Somme de tous les éléments	`np.sum(A)` $\to 22$
`np.prod(A)`	Produit de tous les éléments	`np.prod(A)` $\to 0$
`np.min(A)`	Élément minimum	`np.min(A)` $\to 0$
`np.max(A)`	Élément maximum	`np.max(A)` $\to 6$
`np.mean(A)`	Moyenne de tous les éléments	`np.mean(A)` $\to 2.44$
`np.cumsum(A)`	Sommes cumulées	`np.cumsum(A)` $\to$ `array([1,3,6,6,7,11,16,22,22])`
`np.median(A)`	Médiane de tous les éléments	`np.median(A)` $\to 2$
`np.transpose(A)`	Transposée $= \begin{pmatrix} 1 & 0 & 5 \\ 2 & 1 & 6 \\ 3 & 4 & 0 \end{pmatrix}$	`array([[1,0,5],[2,1,6],[3,4,0]])`
`np.eye(n)`	Matrice identité de taille $n \times n$	`np.eye(3)` $\to$ `array([[1,0,0],[0,1,0],[0,0,1]])`
`np.zeros((n,p))`	Matrice nulle de taille $n \times p$	`np.zeros((2,3))` $\to$ `array([[0, 0, 0],[0, 0, 0]])`
`np.ones((n,p))`	Matrice de taille $n \times p$ remplie de 1	`np.ones((2,3))` $\to \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

On utilisera aussi la bibliothèque `numpy.linalg` pour des opérations plus avancées, qu'on appellera avec

`import numpy.linalg as al`

Fonction	**Effet**
`al.inv(A)`	Si A est inversible, retourne son inverse A^{-1}
`al.rank(A)`	Rang de la matrice A
`al.matrix_power(A, n)`	A^n
`al.solve(A, b)`	Résout le système $Ax = b$ s'il admet une solution
`al.eig(A)`	Valeurs propres et vecteurs propres de A (voir en fin d'année)

5 Exercices

5.1 Vecteurs

Exercice 1 A l'aide du cours, deviner l'action des commandes suivantes. Si besoin, les taper ensuite dans la console dans l'ordre pour vérifier et annoter vos feuilles !

```
x=np.arange(1,10)
x # dans la console, print est inutile
y=np.arange(1,10,2)
y
z=np.linspace(1,10,3)
z
t=np.linspace(0,1,10)
t
a=np.array([1,3,8])
a
np.shape(a)
a[1]
a[1:]
np.min(a)
np.max(a)
np.sum(a)
np.prod(a)
```

Exercice 2 Opérations coefficients par coefficients sur les tableaux. Comme pour l'exercice 1, deviner les résultats avant de taper les instructions dans la console !

```
x=np.array([-1,1,5])
2+x
3*x
np.exp(x)
x*x
x**2
2.**x # point nécessaire pour que 2 soit
    # vu comme un réel et non un entier
1/x
y=np.array([0,-1,-2])
x*y
y/x
C=np.ones(5)
C[2]=0
C
```

Exercice 3 (A vous de jouer) Créer les vecteurs suivants, sans rentrer les coefficients à la main :
a $= [1, 2, \ldots, 99, 100]$, b $= [2, 4, 6, \ldots, 48, 50]$, c $= [20, 19, 18, \ldots, 1]$, d $= [1.5, 2.5, \ldots, 10.5]$,
e = vecteur contenant 50 coefficients allant de 1 à 30 équidistants, f $= (5, 5, \ldots, 5)$ de taille 50

Exercice 4

1. Créer le vecteur x $= [8, 13, 11, 7, 5, 4, 9, 1]$.
2. Ecrire une commande pour récupérer l'élément 8, puis une autre pour récupérer l'élément 7 .
3. Ecrire une commande pour remplacer 4 par 3.
4. Ecrire une commande pour créer le vecteur $\left[e^8, e^{13}, e^{11}, e^7, e^5, e^3, e^9, e\right]$.

Exercice 5 A l'aide de la commande `np.arange(début,fin +1 ,pas)` et SANS boucle for, créer :

1. le tableau de taille 20 qui comporte les 20 premiers entiers non-nuls au carré.
2. (a) une fonction qui prend en entrée un entier naturel n et qui renvoie la valeur de $\sum_{k=1}^{n} k^3$.
   (b) Révisions : comment avait-on fait sans les tableaux, à l'aide de for ? Réécrire la fonction correspondante.
3. (a) une fonction qui prend en entrée un entier naturel n et qui renvoie la valeur de $n!$.
   (b) Révisions : comment avait-on fait sans les tableaux à l'aide de for ? Réécrire la fonction correspondante.
4. le vecteur de taille 10 : $((\text{-1})\ ^1, (-1)^2, \ldots, (-1)^{10})$.
5. le vecteur des nombres de la forme $(-1)^n n^2$, pour $n = 1, .., 20$
6. bonus : une fonction qui prend en entrée deux entiers naturels non nuls n et p et qui renvoie le calcul de $\sum_{k=1}^{n} k^p$.

Exercice 6 (Edhec E 2019) Soit $(u_n)_{n\in\mathbb{N}^*}$ la suite définie pour tout $n \in \mathbb{N}^*$ par $u_n = \dfrac{4^n(n!)^2}{(2n+1)!}$.

On admet que, si t est un vecteur, la commande `np.prod(t)` renvoie le produit des éléments de `t`. Compléter le script suivant afin qu'il permette de calculer et d'afficher la valeur de u_n pour une valeur n entrée par l'utilisateur.

```
n=int(input('entrez une valeur pour n : '))
x=np.arange(1,n+1)
m=2*n+1
y=np.arange(1,m+1)
v=------
w=------
u=------*v**2/w
print(u)
```

Exercice 7 (Suites et Vecteurs) Soit la suite $(u_n)_{n\in\mathbb{N}}$ définie par la relation de récurrence $u_0 = 1$ et $\forall n \in \mathbb{N}^*, u_{n+1} = e^{1/u_n}$.

1. Ecrire une fonction python qui à un entier n associe le réel u_n. (pour vérifier $u_5 \simeq 1.83$)
2. Compléter la fonction précédente afin qu'elle donne en sortie le vecteur $U = (u_0, u_1, u_2, \ldots, u_n)$.

Indication : Introduire le vecteur U via `np.zeros(n+1)`, puis modifier les coefficients de U via une boucle for, cf TPs précédents.

Exercice 8 Reprendre les questions de l'exercice précédent avec la suite u définie par : $u_0 = 2$ et $\forall n \in \mathbb{N}, u_{n+1} = \ln(1 + 2u_n)$. Que peut-on conjecturer sur la convergence de la suite ?

Exercice 9 (Edhec S 2017 (hors question préliminaire))

Question préliminaire : Deviner les affichages des instructions suivantes : $2 * *\mathrm{x}$ et $\mathrm{x} * *2$ avec $\mathrm{x} = \mathrm{np} \cdot \mathrm{array}([1, 2, 3])$

Soit n un entier supérieur ou égal à 1, on considère la fonction f_n définie par : $\forall x \in [0, 1], f_n(x) = \sum_{k=1}^{n} x^k$.

1. (a) Compléter la fonction python suivante pour qu'elle renvoie la valeur de $f_n(x)$ à l'appel de $\mathbf{f(x, n)}$, où x et n sont donnés par l'utilisateur.

```
def f(x,n):
    return np.sum(------)
```

(b) Transformer pour $x \neq 1$, l'expression de $f_n(x)$ puis en déduire une deuxième façon de déclarer **f**, en complétant la déclaration suivante où la fonction est toujours nommée f :

```
def f(x,n):
      if x==1:
          return ------
      else:
          ------
```

2. Montrer que l'équation $f_n(x) = 1$ d'inconnue $x \in [0, 1]$ possède une unique solution a_n dans $[0, 1]$.
3. (a) Montrer que $f_{n+1}(a_n) \geq 1$ et en déduire que la suite $(a_n)_{n\in\mathbb{N}^*}$ est décroissante.
   (b) En déduire que la suite $(a_n)_{n\in\mathbb{N}^*}$ converge.
4. (a) Déterminer a_2 puis vérifier que $0 \leq a_2 < 1$.
   (b) Utiliser les variations de la suite $(a_n)_{n\in\mathbb{N}^*}$ pour établir que $\lim\limits_{n\to+\infty} a_n^{n+1} = 0$.
   (c) En déduire que $\lim\limits_{n\to+\infty} a_n = \dfrac{1}{2}$.
5. On suppose que f_n a été déclarée (voir question 1) et on considère les commandes supplémentaires suivantes :

```
n=int(input('entrer la valeur de n :'))
x=0
while f(x,n)<1:
    x=x+0.001
print(x)
```

Quel est le lien entre le résultat affiché et a_n ?

Exercice 10 (inspiré d'Edhec S 2017 pour les plus rapides uniquement)**

Rappel : si u est un vecteur, la commande `np.cumprod(u)` renvoie un vecteur de même format que u dont le k^{ieme} élément est le produit des k premiers éléments de u.

Compléter la fonction suivante afin qu'elle retourne la variable s contenant la valeur $k!\sum_{i=1}^{k}\frac{x^i}{i!}e^{-x}$:

```
def edhec2017(k,x):
    p=np.prod(np.arange(1,k+1))
    u=------ / ------
    s=p*------*np.exp(-x)
    return s
```

Exercice 11 (Vecteurs et probabilités)

Rappel : la commande rd.randint $(1, 7)$ simule un lancer de dé équilibré. On suppose les bibliothèques numpy et numpy.random importées. Lire (sans exécuter) le programme suivant :

```
n=int(input('entrer un entier n '))
T=np.zeros(6)
for k in range(n):
    i=rd.randint(1,7)
    T[i-1]=T[i-1]+1
print(T)
```

1. Prenons $n = 5$ et supposons que les 5 simulations successives de lancers de dé ont donné les résultats : $2, 1, 6, 1, 1$. Que contient le tableau T au départ ? Comment évolue-t-il ? Que contient-il à la fin ?
2. Retour au cas général : que contient le tableau T à la fin ?
3. Dans le cas où n est grand, donner un équivalent des valeurs contenues dans le tableau T.

5.2 Matrices

Exercice 12 Sans rentrer les cœfficients un à un, déclarer les matrices $A = \begin{pmatrix}5&3&3\\3&5&3\\3&3&5\end{pmatrix}$ et $B = \begin{pmatrix}1&0&0&0\\1&1&0&0\\1&0&1&0\\1&0&0&1\end{pmatrix}$.

Exercice 13 Soit A la matrice définie par $A = \begin{pmatrix}1&0&1\\0&1&0\\0&0&1\end{pmatrix}$.

1. Conjecturer la forme de A^n à l'aide de python.
2. Démontrer alors votre conjecture (on pensera à 2 méthodes).

Exercice 14 Que vaut le produit d'un vecteur colonne $\begin{pmatrix}a_1\\\vdots\\a_n\end{pmatrix}$ par le vecteur ligne $(1 \quad \dots \quad 1)$?

En déduire une ligne de commande créant la matrice $\begin{pmatrix}1&1&\dots&1\\2&2&\dots&2\\\vdots&\vdots&&\vdots\\10&10&\dots&10\end{pmatrix} \in \mathscr{M}_{10}(\mathbb{R})$.

Exercice 15

1. On définit la matrice $A = \begin{pmatrix}1&1&2&0\\1&-1&0&-2\\0&1&1&1\\1&0&1&-1\end{pmatrix}$.
   a) Déclarer la matrice A.
   b) Appliquer l'algorithme du pivot de Gauss à l'aide de `Python` (on effectuera chacune des opérations élémentaires sur `Python`). En déduire le rang.
   c) Vérifier votre résultat à l'aide de la fonction `al.matrix_rank`.
2. On considère à présent la matrice $B = \begin{pmatrix}1&1&0\\1&-1&1\\1&0&1\end{pmatrix}$.
   a) Déclarer la matrice B. Vérifier que B est inversible et calculer son inverse à l'aide de `Python`.

b) Résoudre à l'aide de `Python` le système $\begin{cases} x+y=1 \\ x-y+z=3 \\ x+z=-2 \end{cases}$.

Exercice 16 Écrire une fonction d'en-tête `def trace(A)` prenant en entrée une matrice `A`, et renvoyant la trace de `A` si la matrice entrée est carrée, et un message d'erreur sinon.

Exercice 17 1. a) Établir que, lorsque k est inférieur ou égal à n :

$$\binom{n}{k} = \prod_{i=1}^{k} \left(\frac{n+1}{i} - 1 \right).$$

b) Écrire une fonction d'en-tête `def coeffbin(k,n):` permettant de calculer $\binom{n}{k}$ à partir des vecteurs `np.arange(1,k+1)` et `np.ones(k)`.

c) Retrouver ce résultat avec la commande suivante (qu'on justifiera) :

```
np.cumprod(np.arange(n,0,-1)/np.arange(1,n+1))
```

2. On désire créer une matrice de taille $n \times n$ ($n \geq 1$) dont l'élément de la ligne $i+1$ et de la colonne $j+1$ est égal à $\binom{i}{j}$, avec i et j dans $[\![0, n-1]\!]$.

a) Donner une ligne de commande permettant de créer une matrice C de taille $n \times n$ dont les éléments de la première colonne sont égaux à 1 (les autres éléments étant nuls).

b) Écrire une fonction d'en-tête `def triangle_Pascal(n)` qui remplit la matrice C à l'aide de la formule de Pascal et renvoie ainsi la matrice voulue (triangle de Pascal).

Je remercie Monsieur Mansuy et Madame Fritz dont le document ci-dessus est inspiré.