TP de Python numéro 8 : Matrices et le type `array` de `numpy`

*Semaine du **6 février**.*

Pensez, lors des exercices, à bien vérifier vos fonctions sur des exemples variés !

Dans tous les énoncés et exemples de ce TP, on suppose `numpy` importé via la commande :

```
import numpy as np
```

I. Définition de matrices avec `numpy`

1. Modélisation informatique des matrices

Pour modéliser une matrice en informatique, on dit qu'une matrice est représentée par la liste de ses lignes, elles-mêmes représentées comme des listes.

Par exemple, **en première approche** (ceci sera immédiatement changé), la matrice $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ est donnée par la liste :

```
A=[ [1, 2], [3, 4] ]
```

Le problème si on s'arrêtait ici, c'est que les opérations dont on dispose sur les listes ne correspondent pas du tout à celles sur les matrices. Par exemple, la somme de deux listes est leur concaténation :

```
>>> A=[ [1, 2], [3, 4] ]

>>> B=[ [0, 0], [1, 1] ]

>>> A+B
[[1, 2], [3, 4], [0, 0], [1, 1]]
```

$$\textit{La somme } \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \textit{ n'est pas du tout } \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 0 & 0 \\ 1 & 1 \end{pmatrix} !$$

Afin de représenter les matrices avec Python en ayant accès aux opérations matricielles et à d'autres outils pratiques, on utilisera le type `array` de `numpy`, qui vient avec une multitude d'outils que nous allons explorer.

On rappelle que dans ce TP, `numpy` est importé avec l'alias `np`.

Le type `array` de `numpy`

- Le type `array` de `numpy` est désigné en Python par :
  `numpy.ndarray`
  (`nd` *stands for : "n-dimensionnal".*)
- Soit L une liste de liste représentant une matrice au sens précédent. Alors, la commande
  `A=np.array(L)`
  créer une nouvelle variable `A` contenant l'objet de type `numpy.ndarray` donné par la liste `L`.

Exemple 1.

```
>>> import numpy as np

>>> L=[ [1, 2], [3, 4]]

>>> A=np.array(L)

>>> A
array([[1, 2],
       [3, 4]])

>>> type(A)
<class 'numpy.ndarray'>
```

Pour définir une matrice en Python, on peut donc entrer ses coefficicents ligne par ligne dans une liste de liste, puis transformer cette liste de liste en objet de type `array` avec la commande `np.array`.

Remarque. Pour que `np.array` ait l'effet voulu, il faut bien entrer une liste de liste représentant une matrice, c'est-à-dire pour laquelle toutes les "lignes" sont de la même longueur.

Exercice 2. 1. Définir en Python les matrices $A = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix}$, $B = \begin{pmatrix} 1 & -1 & 0 \\ 3 & 2 & 3 \end{pmatrix}$ et $C = \begin{pmatrix} 1 & 2 \\ 3 & 1 \\ 4 & -6 \end{pmatrix}$.

2. Afficher ces matrices avec la commande `print` et constater la particularité de l'affichage.

2. Taille d'une matrice, accès aux coefficients, aux lignes et aux colonnes

Taille d'une matrice : la fonction `shape`

Soit `A` une variable de type `array`.
Alors, la commande `np.shape(A)` renvoie le couple `(n,p)` formé du nombre `n` de lignes de `A` et du nombre `p` de colonnes de `A`.

Par exemple, pour $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$, $B = \begin{pmatrix} 2 & 3 & 2 & 3 \end{pmatrix}$ et $C = \begin{pmatrix} 1 \\ 3 \\ 5 \end{pmatrix}$:

```
A=np.array([ [1, 2], [3, 4], [5, 6] ])
n, p = np.shape(A)
print(n) # Affiche 2
print(p) # Affiche 3

B=np.array([ [ 2, 3, 2, 3 ] ])
n, p = np.shape(B)
print(n) # Affiche 1 : B est une matrice ligne...
print(p) # Affiche 4 : ...de taille 4

C=np.array([ [1], [3], [5] ])
print(np.shape(C)) # Affiche (3,1) : C est une matrice colonne de taille 3
```

Remarque. Attention aux matrices lignes.

- Pour saisir une matrice ligne comme $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$, on doit bien **mettre deux crochets** dans la commande :

  `np.array( [ [1, 2, 3] ] )`

  Autrement dit, on rentre bien un matrice en donnant la liste de ses lignes, même si celle-ci n'a qu'une ligne.

- Soit `L=[`x_1`,...,`x_n`]` une liste de nombres.

  Alors, la commande `X=np.array(L)` créée une variable `X` de type `array` représentant **le vecteur** $(x_1, ..., x_n)$ (un vecteur n'est ici rien d'autre qu'un élément de $\mathbb{R}^n$ pour un certain $n \in \mathbb{N}^*$).

  La différence avec la notion de matrice ligne est minime mais existe. Par exemple, la taille d'un vecteur n'est pas un couple d'entiers mais un 1-uplet formé d'un entier, ce qui peut provoquer un problème d'exécution dans un code utilisant `np.shape` écrit pour des matrices. Par exemple :

```
X=np.array([ 1, 2, 3])
print(np.shape(X)) # affiche (3,) (et non (1,3))
n,p=np.shape(X) # provoque une erreur
```

Exercice 3. Écrire le code d'une fonction Python, nommée `test_produit`, prenant en paramètres deux variables de type `array` A et B représentant des matrices et renvoyant en sortie `True` si le produit matriciel AB est correctement défini et `False` sinon.

Accès aux coefficients d'une matrice, aux lignes et aux colonnes (première version)

Soit `A` une variable de type `array` représentant une matrice. Notons (n,p) la taille de la matrice représentée par `A`. Alors :

- Pour tout `(i,j)` $\in [\![0,n-1]\!] \times [\![0,p-1]\!]$, les commandes `A[i,j]` et `A[i][j]` renvoient le coefficient d'indice `(i+1,j+1)` de la matrice représentée par `A`.
- Pour tout `i` $\in [\![0,n-1]\!]$, les commandes `A[i]` et `A[i,:]` renvoient **le vecteur** donné par la `i+1`-ième ligne de la matrice représentée par `A`.
- Pour tout `j` $\in [\![0,p-1]\!]$, les commandes `A[j]` et `A[j,:]` renvoient **le vecteur** donné par la `j+1`-ième colonne de la matrice représentée par `A`.

Remarque. Attention, comme pour les listes, l'indexation des coefficients commence à 0 en Python ! C'est une source d'erreur très courante au début, prenez vite la bonne habitude d'y penser.

Exemple 4. Anticiper les affichages suivants, puis vérifier avec Python (avec la matrice $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$):

```
A=np.array([ [1, 2, 3], [4, 5, 6]])
print(A[1,1]) # Affiche ...
print(A[0,2]) # Affiche
print(A[1,:]) # Affiche
print(A[:,2]) # Affiche
print(A[:,3]) # Affiche
```

Indice d'une variable de type `array`

A partir de maintenant, dans ce TP, si `A` est une variable de type `array` représentant une matrice, **l'indice** d'un coefficient, d'une ligne ou d'une colonne de l'objet `A` désignera son indice en Python.
Par exemple, la première ligne (au sens classique) d'un objet `A` de type `array` sera désignée comme sa ligne d'indice 0. Le coefficient d'indice $(1,2)$ de la matrice représentée par `A` sera désigné comme le coefficient d'incide `(0,1)` de `A`.

Accès aux lignes et aux colonnes (deuxième version), extraction de sous-matrices

Soit `A` une variable de type `array` représentant une matrice, dont on note (n,p) la taille.
Soient `d,f` deux éléments de $[\![0,n-1]\!]$ tels que `d<f`.
Soient `d',f'` deux éléments de $[\![0,p-1]\!]$ tels que `d'<f'`.

- La commande `A[d:f, :]` renvoie l'objet de type `array` obtenu à partir de `A` en gardant uniquement les lignes de `A` de l'indice `d` à l'indice `f-1`.
- La commande `A[:, d':f']` renvoie l'objet de type `array` obtenu à partir de `A` en gardant uniquement les colonnes de `A` de l'indice `d'` à l'indice `f'-1`.
- La commande `A[d:f, d':f']` renvoie l'objet de type `array` obtenu à partir de `A` en gardant uniquement les lignes de `A` de l'indice `d` à l'indice `f-1` et les colonnes de `A` de l'indice `d'` à l'indice `f'-1`.
- Par conséquent, la commande `A[:, d:d+1]` renvoie la ligne d'indice `d` de `A` en tant qu'objet de type `array` représentant une matrice.
- Par conséquent, la commande `A[:, d':d'+1]` renvoie la colonne d'indice `d'` de `A` en tant qu'objet de type `array` représentant une matrice.

Exemple 5. A partir de la matrice $A = \begin{pmatrix} 2 & 4 & 6 \\ 2 & 4 & 8 \\ 2 & 4 & 1 \end{pmatrix}$, on peut définir en Python les matrices $B = \begin{pmatrix} 2 & 4 & 8 \end{pmatrix}$ et $C = \begin{pmatrix} 4 & 6 \\ 4 & 8 \end{pmatrix}$ de la manière suivante.

```
A= np.array([[2, 4, 6], [2, 4, 8], [2, 4, 1]])
B= A[1:2,:] # B est la ligne d'indice 1 de A, en tant que matrice
C= A[0:2,1:3] # C est obtenue à partir de A en gardant les lignes d'indice 0 et 1 et
    colonnes d'indices 1 et 2.
```

3. Modification d'une matrice, copie d'une matrice

Comme pour les listes, les coefficients d'un objet de type `array` sont des variables. On peut donc les modifier avec `=`.

Au début de ce code (après la première ligne), la variable `A` représente la matrice $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

```
A=np.array([ [1, 2], [3, 4] ])
print(A) # Affiche [[1 2]
         #          [3 4]]



A[0,0]=3 # Changement du coefficient d'indice (0,0)
A[1,0]=6 # Changement du coefficient d'indice (1,0)
print(A) # Affiche [[3 2]
         #          [6 4]]
```

À la fin de ce code, la variable `A` représente la matrice $\begin{pmatrix} 3 & 2 \\ 6 & 4 \end{pmatrix}$.

Comme pour les listes, l'utilisation du symbole `=` pour définir une nouvelle variable contenant une matrice lie ces matrices.

```
A=np.array([[1,2], [3,4] ])
B=A
B[0,0]=10
print(B) #Affiche [[10  2]
         #         [ 3  4]]
print(A) #Affiche [[10  2]
         #         [ 3  4]]
```

Pour copier une matrice **en une matrice indépendante**, on peut utiliser la commande `np.copy`

```
A=np.array([[1,2], [3,4] ])
B=np.copy(A)
B[0,0]=10
print(B) #Affiche [[10  2]
         #         [ 3  4]]
print(A) #Affiche [[1  2]
         #         [ 3  4]]
```

4. Égalité de matrices

Attention, **le test d'égalité** `==` entre deux matrices fonctionne de manière très particulière.

Exemple 6. Recopier et exécuter dans l'invite de commande :

```
>>> A=np.array([[1,2], [3,4]])
>>> B=np.array([[2,1], [3,4]])
>>> A==B
```

Qu'est-ce-qui est renvoyé par ce test d'égalité ?

Exercice 7. Écrire le code d'une fonction `Python` d'entête `def EgaliteMatrices(A,B):` prenant en entrée deux matrices `A` et `B` (de type `array`), et renvoyant `True` si ces matrices sont égales, et `False` sinon. *On rappelle que deux matrice sont égales si et seulement si elles ont la même taille et les mêmes familles de coefficients.*

On pourra utiliser les opérateurs de comparaison (`==, >=, >...`) en ayant remarqué le caractère particulier de la valeur renvoyée.

Opérateurs booléens et matrices

Soient `A` et `B` deux objets de type `array` de même taille (n, p).

- La commande `A==B` renvoie l'objet de type `array` de taille (n, p) dont le coefficient d'indice `(i,j)` est le renvoi de :
  `A[i,j] == B[i,j]`
  (pour tout `(i,j)` $\in [\![0, n-1]\!] \times [\![0, p-1]\!]$.)
- Si `k` est une variable numérique, la commande `A>k` renvoie l'objet de type `array` de taille (n, p) dont le coefficient d'indice `(i,j)` est le renvoi de :
  `A[i,j] > k`
  (pour tout `(i,j)` $\in [\![0, n-1]\!] \times [\![0, p-1]\!]$.)
- Le point précédent est valable à l'identique avec les autres opérateurs de comparaison (`==`,`>=`, `<`, `<=`) entre une matrice et un nombre.

Exercice 8. Écrire le code d'une fonction Python, nommée `coefficients_pairs`, prenant en paramètres une matrice A à coefficients entiers relatifs et renvoyant en sortie `True` si tous les coefficients de A sont pairs et `False` sinon.

5. Créations de matrices

Il existe de nombreuses commandes permettant de définir des matrices sans devoir saisir les coefficients à la main.

Matrices remarquables

Soient `n` et `p` deux variables de type `int`.

- `np.zeros( (n,p) )` renvoie la matrice nulle de taille `(n,p)` .
- `np.ones( (n,p) )` renvoie la matrice de taille `(n,p)` dont tous les coefficients valent 1.
- `np.eye(n)` (ou `np.identity(n)`) renvoie la matrice identité de taille `n`.
- `np.eye(n,p)` renvoie la matrice de taille `(n,p)` dont tous les coefficients sont nuls sauf les coefficients diagonaux qui valent 1.
- `np.zeros(n)` renvoie le vecteur de longueur `n` dont tous les coefficients sont nuls.
- `np.ones(n)` renvoie le vecteur de longueur `n` dont tous les coefficients sont égaux à 1.

Remarque. Attention à la présence des double-parenthèses dans les commandes `np.zeros((n,p))` et `np.ones((n,p))`. Les oublier provoquera une erreur. On retiendra que ces commandes prennent en argument `une taille`, et qu'une taille est un couple pour une matrice.

Exemple 9.

```
A = np.zeros(3)
print(A) # Affiche [0. 0. 0.]

B = np.zeros((2,2))
print(B) # Affiche [[0. 0.]
         #          [0. 0.]]

C = np.ones(5)
print(C) # Affiche [1. 1. 1. 1. 1.]

D = np.ones((3,3))
print(D) # Affiche [[1. 1. 1.]
         #          [1. 1. 1.]
         #          [1. 1. 1.]]

E = np.eye(3)
print(E) # Affiche [[1. 0. 0.]
         #          [0. 1. 0.]
         #          [0. 0. 1.]]

F = np.eye(2,4)
print(F) # Affiche [[1. 0. 0. 0.]
         #          [0. 1. 0. 0.]]
```

Remarque. Les commandes `np.linspace` et `np.arange` rencontrées lors du TP définissent des vecteurs de type `array`. Ça peut être pratique dans certaines situations. Par exemple :

`X=np.arange(0,2,0.5)` a le même effet que `X = np.array([0, 0.5, 1, 1.5])`.

6. Une stratégie courante pour définir des matrices

Très souvent, pour définir une matrice donnée en Python :

- On définit une matrice `A` de la bonne taille dont les coefficients sont tous nuls avec la commande

  `A=np.zeros((n,p))`

- On affecte un à un les coefficients de `A` à l'aide de boucles.

Exercice 10. On pose, pour tous entiers i et j, $a_{i,j} = 2^i 3^{j+1}$. Écrire un code Python permettant de définir la matrice $(a_{i,j})_{(i,j)\in[\![1,20]\!]\times[\![1,10]\!]}$ en Python.

Exercice 11. Dans cet exercice, on suppose que les fonctionnalités de la partie II ne sont pas connues.
Écrire le code d'une fonction Python, nommée `somme_matrices`, prenant en paramètres deux matrices A et B de même taille et renvoyant en sortie la matrice $A + B$.

Exercice 12. Dans cet exercice, on suppose que les fonctionnalités de la partie II ne sont pas connues.
Écrire le code d'une fonction Python, nommée `transpose_matrice`, prenant en paramètre une matrice A et renvoyant en sortie la matrice ${}^tA$.

Exercice 13. Dans cet exercice, on suppose que les fonctionnalités de la partie II ne sont pas connues.
Écrire le code d'une fonction Python, nommée `produit_matrices`, prenant en paramètres deux matrices (Python) A et B telles que le produit AB est correctement défini et renvoyant en sortie la matrice AB.

Exercice 14. Écrire le code d'une fonction Python, nommée `test_commutation`, prenant en paramètres deux matrices carrés A et B de même taille et renvoyant en sortie `True` si les matrices A et B commutent et `False` sinon.

II. Opérations sur les matrices

On va maintenant voir les opérations disponibles sur les objets de type `array`.

A partir de maintenant, on parlera simplement de matrice pour désigner des variables Python de type `array` représentant des matrices.

1. Opérations coefficients par coefficients

Opérations arithmétiques de base entre matrices

Soient `A` et `B` deux matrices **de même taille**.
Alors, les opérations arithmétiques usuelles : `+, -, *, /, **` s'appliquent coefficients par coefficients aux matrices avec les commandes :
`A+B, A-B, A*B, A/B, A**B,`
à condition que tous les calculs intervenant soient correctement définis.

Remarque. Dans certains cas, il peut y avoir des problèmes si `A` et `B` ont des coefficients de type `int`. On travaille généralement avec des coefficients de type `float` pour éviter ça, même si ceux-ci sont des entiers.

Exemple 15. Vérifier la compréhension du point précédent en observant les valeurs affichées par les commandes ci-dessous.

```
A=np.array([ [2.0, 1.0], [0.0, 3.0] ])
B=np.array([ [-1.0, 1.0], [4.0, 2.0] ])

print(A+B)
print(A-B)
print(A*B)
print(A/B)
print(A**B)
```

Remarque. En particulier, le symbole de multiplication `*` **n'est pas le produit matriciel**. Par contre, la somme `+` et la différence `-` donnent bien la somme et la différence matricielle.

Produit d'une matrice par un réel

Soit `A` une matrice et `x` un réel (de type `float` ou `int`). Alors, la commande :

`x*A`

renvoie la matrice obtenue en multipliant chaque coefficient de `A` par `x`.

Remarque. On a bien l'opération $\cdot$ de multiplication d'une matrice par un réel avec le symbole `*`.

Exemple 16. Vérifier ce qui précède en exécutant le code suivant.

```
A = np.array([[2.0,0.0],[1.0,3.0]])
B = 3*A
C = -2*np.eye(3)
print(A)
print(B)
print(C)
```

Somme d'une matrice et d'un réel

Soit `A` une matrice et `x` un réel (de type `float` ou `int`). Alors, la commande :

`A+x`

renvoie la matrice obtenue en ajoutant `x` à chaque coefficient de `A`.

Remarque. Attention, n'écrivez pas n'importe quoi dans vos copies : cette opération n'existe toujours pas en mathématiques.

Exemple 17. Vérifier ce qui précède en exécutant le code suivant.

```
A = np.array([[1,3],[-2,5]])
print(A+2)
print(A-5)
print(3*A+1)
```

Fonctions usuelles de `numpy`

Soit `A` une matrice. Alors, l'application des fonctions usuelles de `numpy` (`p.abs`, `np.sqrt`, `np.exp`, `np.log`, `np.floor`) à la matrice `A` renvoie la matrice obtenue en appliquant ces fonctions à chaque coefficients de `A`.

Exemple 18. Vérifier ce qui précède en exécutant le code suivant.

```
A=np.array([ [1.1, 2.3], [3.5, 4.0] ])
print(np.sqrt(A))
print(np.exp(A))
print(np.floor(A))
```

On peut également, à partir d'une fonction `f` définie par vos soins, créer une version de `f` qui s'applique coefficient par coefficient à une matrice.

Vectorisation de fonctions : `np.vectorize`

Soit `f` une fonction informatique représentant une fonction réelle de la variable réelle.
Alors, la commande `g = np.vectorize(f)`
créer une fonction `g` qui applique la fonction `f` coefficient par coefficient à une matrice donnée en entrée : l'appel de `g(A)` renvoie la matrice obtenue en remplaçant chaque coefficient `a` de `A` par `f(a)`, et ce pour toute matrice `A`.

Exemple 19. Vérifier ce qui précède en complétant et en exécutant le code suivant.

```
def f(x):
    # f(x) renvoi x si x >0 et 0 sinon.
    if x >0:
        return(x)
    return(0)

A=np.array([ [6, -2], [-3, 1] ])
g=np.vectorize(f)
print(g(A))
```

2. Produit matriciel et transposition

Produit matriciel

Soient `A` et `B` deux matrices Python représentant des matrices A et B telles que le produit AB soit bien défini.
Alors, la commande

`np.dot(A,B)`

renvoie la matrice Python représentant la matrice AB.

Transposition

Soit `A` une matrice Python.
Alors, la commande

`np.transpose(A)`

renvoie la transposée de `A`.

Exemple 20. Vérifier ce qui précède en exécutant le code suivant.

```
A=np.array([[ 0, 1], [0, 0] ])

print("Transposée : ", np.transpose(A))
print("Carré de A : ", np.dot(A,A))

B=np.array([ [1, 1], [1, 1] ])
print(np.dot(A,B))
print(np.dot(B,A))
```

Exercice 21. Dans cet exercice, on suppose que les fonctionnalités de la partie suivante ne sont pas connues. Écrire le code d'une fonction Python, nommée `puissance_matrice`, prenant en paramètres une matrice carrée A et un entier naturel n et renvoyant en sortie la matrice A^n.

Exercice 22.

1. Écrire le code d'une fonction Python, nommée `est_symetrique`, prenant en paramètre une matrice A et renvoyant en sortie `True` si la matrice A est symétrique et `False` sinon.
2. Écrire le code d'une fonction Python, nommée `est_antisymetrique`, prenant en paramètre une matrice A et renvoyant en sortie `True` si la matrice A est antisymétrique et `False` sinon.

3. Plus d'opérations avec le sous-module `numpy.linalg` de `numpy`

Le sous module `numpy.linagl` de `numpy` contient des commandes permettant des opérations plus avancées sur les matrices (et vous en verrez bien d'autres l'année prochaine).

Cette année, on utilisera deux commandes :

- Une commande permettant de calculer des puissances d'une matrice carrée (plutôt que d'utiliser `np.dot` de manière répétée,
- Une commande permettant de calculer l'inverse d'une matrice inversible.

Import du sous-module numpy.linalg

Dans les encadrés ci-dessous, on supposera le module `numpy.linalg` importé à l'aide de la commande :

```
import numpy.linalg as al
```

Puissances d'une matrice carrée

Soit `A` une matrice carrée (en Python).
Alors, pour toute variable `n` de type `int`, la commande
`al.matrix_power(A,n)`
renvoie la puissance `n`-ième $\texttt{A}^{\texttt{n}}$ de la matrice `A`.

Exemple 23. Utiliser cette commande pour afficher les puissances A^k de la matrice $A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$,

pour $k \in [\![0, 5]\!]$. Que remarquez-vous ?

Inverse d'une matrice inversible

Soit `A` une matrice Python représentant une matrice inversible.
Alors, l'inverse de `A` est renvoyé par la commande :

```
al.inv(A)
```

Exemple 24. Donner l'inverse de $A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$.

III. Exercices

Exercice 25. Écrire le code d'une fonction Python, nommée `somme_puissances_matrice`, prenant en paramètres une matrice carrée A telle que $A \in \mathcal{M}_n(\mathbb{R})$ et renvoyant en sortie la matrice $\mathrm{I}_n + A + \cdots + A^n$.

On pourra observer que pour tout $k \in [\![1, n]\!]$, $\mathrm{I}_n + A + \cdots + A^k = \mathrm{I}_n + A\left(\mathrm{I}_n + A + \cdots + A^{k-1}\right)$.

Exercice 26. Dans cet exercice, on s'intéresse à la création informatique du triangle de Pascal. Les matrices à construire ne devront pas l'être en saisissant à la main les coefficients un par un.

1. Écrire un code Python permettant de définir la matrice $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$.
2. Écrire un code Python permettant de définir la matrice $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 1 & 3 & 3 & 1 & 0 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$. On utilisera la même méthode que celle qu'on utilise pour écrire le triangle de Pascal à la main.
3. Écrire le code d'une fonction Python, nommée `triangle_Pascal`, prenant en paramètre un entier naturel `n` et renvoyant en sortie la matrice $M \in \mathcal{M}_{n+1}(\mathbb{R})$ dont les coefficients sont ceux du triangle de Pascal, de $\binom{0}{0}$ jusqu'à $\binom{n}{n}$. Tester ensuite cette fonction pour n = 10.
4. Écrire le code d'une fonction Python, nommée VdM, prenant en paramètre un entier naturel `n` et renvoyant en sortie la valeur de la somme $\displaystyle\sum_{k=0}^{n} \binom{n}{k}^2$.

5. À l'aide de nombreux tests effectués grâce à cette fonction, conjecture, puis démontrer, une formule explicite donnant la valeur de $\sum_{k=0}^{n} \binom{n}{k}^2$ en fonction de n, pour tout $n \in \mathbb{N}$.

Exercice 27. Dans cet exercice, on s'intéresse à l'implémentation de l'algorithme du pivot de Gauss.

Soit $A = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 2 & 0 & -2 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & 1 & 1 \\ -3 & 1 & 3 \\ 3 & -3 & -3 \end{pmatrix}$.

1. À l'aide de Python, effectuer les opérations élémentaires suivantes sur la matrice A sans redéfinir à la main les coefficients un à un :
   - échanger les lignes L_1 et L_3.
   - puis remplacer la ligne L_2 par $2L_2 + L_1$
   - puis remplacer la ligne L_3 par $2L_3 + L_2$

   Afficher la matrice obtenue : que peut-on conclure?

2. Écrire le code d'une fonction Python, nommée `multiplie_ligne`, prenant en paramètres une matrice A, un nombre réel a et un entier naturel i et renvoyant en sortie la matrice obtenue à partir de A en multipliant terme à terme par a tous les coefficients de la ligne d'indice i.

3. Écrire le code d'une fonction Python, nommée `echange_lignes`, prenant en paramètres une matrice A et deux entiers naturels i et j et renvoyant en sortie la matrice obtenue à partir de A en échangeant les lignes d'indices i et j.

4. Écrire le code d'une fonction Python, nommée `combinaison_lignes`, prenant en paramètres une matrice A, deux nombres réels a et b et deux entiers naturels i et j et renvoyant en sortie la matrice obtenue à partir de A en remplaçant L_i par la combinaison $aL_i + bL_j$.

5. Tester les fonctions définies ci-dessus pour effectuer les opérations élémentaires suivantes sur la matrice B :
   - $L_2 \longleftrightarrow L_3$
   - $L_2 \longleftarrow L_2 - 3L_1$
   - $L_3 \longleftarrow L_3 + 3L_1$
   - $L_3 \longleftarrow 6L_3 + 4L_2$

   Faire afficher la matrice obtenue : la matrice B est-elle inversible?

6. Écrire le code d'une fonction Python, nommée `pivot_Gauss`, prenant en paramètres une matrice A et renvoyant en sortie la matrice obtenue par application de l'algorithme du pivot de Gauss au système linéaire associé à A jusqu'à le mettre sous forme triangulaire.

   Tester cette fonction avec la matrice $C = \begin{pmatrix} 6 & -3 & 2 & -1 \\ -5 & 2 & -4 & 1 \\ 2 & -1 & 3 & -1 \\ -1 & 1 & -2 & 1 \end{pmatrix}$.