

Les bibliothèques en Python

1	Le module <code>numpy</code> de calcul mathématique	2
1.1	Les fonctions usuelles	2
1.2	Les vecteurs	3
1.3	Les matrices	5
2	Le module <code>numpy.linalg</code> de calcul d'algèbre linéaire	7
3	Le module <code>matplotlib</code> de représentation graphique	8
4	Exercices	9

Compétences attendues.

- ✓ Importer une bibliothèque.
- ✓ Création et opérations sur les vecteurs et les matrices.
- ✓ Tracer la courbe représentative d'une fonction.

Liste des commandes Python exigibles aux concours.

- Importation d'une bibliothèque : `from ... import *`, `import ... as ...`
- Dans la librairie `numpy` :
 - Fonctions et constantes usuelles : `np.exp`, `np.log`, `np.sin`, `np.cos`, `np.sqrt`, `np.abs`, `np.floor`, `np.e`, `np.pi`
 - Création de vecteurs et matrices : `np.array`, `np.zeros`, `np.ones`, `np.eye`, `np.linspace`, `np.arange`
 - Opérations sur les matrices : `+`, `-`, `*`, `/`, `**`, `==`, `>`, `<`, `>=`, `<=`, `!=`, `np.shape`, `np.dot`, `np.transpose`, `np.sum`, `np.mean`, `np.min`, `np.max`, `np.cumsum`
- Dans la librairie `numpy.linalg` : `al.inv`, `al.rank`, `al.matrix_power`, `al.solve`, `al.eig`
- Dans la librairie `matplotlib.pyplot` : `plt.plot`, `plt.show`

Mathieu Mansuy

Professeur en ECG deuxième année spécialité mathématiques approfondies au Lycée Louis Pergaud (Besançon)

Page personnelle : mathieu-mansuy.fr/

E-mail : mathieu.mansuy@ac-besancon.fr

En préambule

Nous avons présenté au TP1 les opérations sur les flottants et les entiers. Pour aller plus loin dans les calculs, on peut espérer avoir accès aux fonctions mathématiques habituelles, telles que cosinus ou exponentielle. Essayons.

```
>>> cos(1.0)

Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'cos' is not defined
```

Cela ne fonctionne donc pas directement ! Et de fait, les fonctions intégrées au langage sont relativement peu nombreuses : ce sont seulement celles qui sont susceptibles d'être utilisées très fréquemment. Les autres sont regroupées dans des fichiers séparés que l'on appelle des *modules*.

Les modules sont des fichiers qui regroupent des ensembles de fonctions. Il en existe un grand nombre pré-programmés qui sont fournis d'office avec Python. Souvent, on essaie de regrouper dans un même module des fonctions apparentées. On parle alors de *bibliothèques*.

Pour bénéficier de ces *bibliothèques*, **il faut les importer**. On utilise pour cela les instructions ci-dessous.

Définition.

Pour importer une bibliothèque, on pourra utiliser la commande :

- `from nom de la bibliothèque import *`
Ceci signifie que l'on importe la totalité des fonctionnalités de la librairie.
- `import nom de la bibliothèque as raccourci`
Ceci signifie que les objets de la bibliothèque pourront être utilisés précédés de ce qui a été choisi pour raccourci.

Voici la liste des bibliothèques au programme de l'ECG :

- `numpy` (raccourci `np`) : dédiée au calcul mathématique, elle donne accès aux principales fonctions mathématiques, aux constantes e et π , et elle permet la création de vecteurs et de matrices.
- `numpy.linalg` (raccourci `al`) : dédiée aux calculs d'algèbre linéaire.
- `numpy.random` (raccourci `rd`) : dédiée à la génération de nombres aléatoires, elle permet de simuler des variables aléatoires.
- `matplotlib.pyplot` (raccourci `plt`) : dédiée au tracé de courbes, de surfaces, de graphiques.

Pour importer la bibliothèque `numpy`, on peut ainsi utiliser au choix l'une des commandes suivantes :

```
from numpy import *   ou   import numpy as np
```

Dans la seconde instruction, `np` est un diminutif pour désigner la bibliothèque `numpy` (on aurait pu choisir autre chose). Si on utilise cette instruction, toutes les commandes seront précédées du préfixe `np`.

Remarque. On peut penser qu'il est donc plus simple d'utiliser la première instruction, mais il peut y avoir conflit si certaines fonctions appartenant à des bibliothèques différentes portent le même nom. On privilégiera donc dans la suite l'importation de bibliothèques à l'aide de diminutifs.

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

1 Le module `numpy` de calcul mathématique

1.1 Les fonctions usuelles

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.

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. Exécuter les instructions suivantes dans la console.

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

1.2 Les vecteurs

Définition.

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. Exécuter 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.

- 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. Exécuter 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)
```

Propriété 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. Exécuter les instructions suivantes dans la console.

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

Propriété 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. Exécuter 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).

1.3 Les matrices

Définition.

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

$$\text{np.array}([[a_{1,1}, \dots, a_{1,p}], \dots, [a_{n,1}, \dots, a_{n,p}]])$$

Exemple. Exécuter les instructions suivantes dans la console :

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

Définition.

- `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.

Propriété 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. Exécuter 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()`.

Propriété 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. Exécuter les instructions suivantes dans la console :

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

Propriété 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
```

Propriété 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).

Propriété 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)`).

Propriété 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`.

Propriété 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)
```

2 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.

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
```

Propriété 10 (Nouvelles commandes matricielles)

On suppose que l'on a déclaré une matrice A .

- La commande `al.inv(A)` renvoie l'inverse de A (si 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 .

Propriété 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.

3 Le module `matplotlib` de représentation graphique

La bibliothèque `matplotlib.pyplot` (plus précisément le sous-module `pyplot` du module `matplotlib`) est dédiée aux représentations graphiques.

Définition.

On importe la bibliothèque `matplotlib.pyplot` en écrivant l'une ou l'autre des instructions suivantes (on privilégiera la deuxième) :

```
from matplotlib.pyplot import *   ou   import matplotlib.pyplot as plt
```

On suppose que x contient le vecteur (x_1, \dots, x_n) et que y contient le vecteur (y_1, \dots, y_n) . Ces vecteurs définissent des points M_1, \dots, M_n de coordonnées respectives $(x_1, y_1), \dots, (x_n, y_n)$. Si on relie les points M_1, \dots, M_n , on parle de ligne brisée. Si les points ne sont pas reliés, on parle de nuage de points.

Définition.

- La commande `plt.plot(x,y)` permet de tracer la ligne brisée reliant les points M_1, \dots, M_n .
- La commande `plt.plot(x,y,'x')` permet de tracer le nuage de points M_1, \dots, M_n , les points étant matérialisés par des croix.

Exemple. Exécuter les instructions suivantes dans la console.

```
>>> x = np.arange(-2,3); y = np.array([3,-1,1,0,1])
>>> plt.plot(x,y)
>>> plt.show()
>>> plt.plot(x,y,'x')
>>> plt.show()
```

Remarque. Sur la plupart des versions de Python, l'affichage se fait automatiquement dans une fenêtre graphique. Selon la distribution de Python utilisée, il est parfois nécessaire d'ajouter l'instruction `plt.show()`.

Propriété 12

Soit f une fonction définie sur un intervalle $[a, b]$ avec $a < b$.

On définit $x = \text{np.linspace}(a, b, n)$ avec n grand. On note x_0, \dots, x_n les composantes du vecteur x et on déclare un vecteur y dont les composantes sont $f(x_0), \dots, f(x_n)$.

Pour représenter la courbe de f , on écrit : `plt.plot(x, y)`.

Exemple. Exécuter les instructions suivantes dans la console.

```
>>> x = np.linspace(-6,6,100) ; y = np.cos(x)
>>> plt.plot(x,y)
>>> plt.show()
```

Remarques. Voici quelques commandes et options associées à la fonction `plot` (voir l'aide pour davantage de précisions) :

- *Superposition ou non de figures.* Sans intervention de notre part, Python superposera les figures dans une même fenêtre graphique. Pour l'éviter, on peut utiliser la commande `plt.figure(n)` où n désigne la n -ème fenêtre graphique à ouvrir. Cette commande doit précéder `plt.plot(x, y)` pour préciser à Python dans quelle fenêtre il doit tracer la figure.
- *Effacer le contenu d'une figure.* `plt.clf()`
- *Axes.* `plt.axis("scaled")` permet de représenter dans un repère orthonormé.
`plt.axis([xmin, xmax, ymin, ymax])` permet de choisir les bornes de la fenêtre d'affichage.
`plt.grid()` permet de rajouter une grille de fond sur votre figure.
- *Couleur.* `plt.plot(x, y, 'r')` si l'on veut un tracé en rouge, 'b' pour le bleu, 'g' pour le vert, 'k' pour le noir.
- *Légendes.* `plt.plot(x, y, label="Cf")` pour associer une étiquette à un tracé.
`plt.legend(loc="upper right")` pour décider de la localisation de la légende sur la figure.
`plt.xlabel("label pour les abscisses")` et `plt.ylabel("label pour les ordonnées")` pour légènder les axes.
`plt.title("Titre de graphe")` pour ajouter un titre au graphe.

4 Exercices

Exercice 1 (★)

Sans rentrer les coefficients 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 2 (★)

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 $\begin{pmatrix} 1 & 1 & \dots & 1 \\ 2 & 2 & \dots & 2 \\ \vdots & \vdots & & \vdots \\ 10 & 10 & \dots & 10 \end{pmatrix} \in \mathcal{M}_{10}(\mathbb{R})$.

Exercice 3 (★)

1. On définit la matrice $A = \begin{pmatrix} 1 & 1 & 2 & 0 \\ 1 & -1 & 0 & -2 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & -1 \end{pmatrix}$.

- Déclarer la matrice A .
- Appliquer l'algorithme du pivot de Gauss à l'aide de `Python` (on effectuera chacune des opérations élémentaires sur `Python`). En déduire le rang.
- Vérifier votre résultat à l'aide de la fonction `al.matrix_rank`.

2. On considère à présent la matrice $B = \begin{pmatrix} 1 & 1 & 0 \\ 1 & -1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$.

- Déclarer la matrice B . Vérifier que B est inversible et calculer son inverse à l'aide de `Python`.

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

Exercice 4 (★★)

1. (a) En une seule ligne de commande, créer le vecteur $x = \left(1, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \dots, \frac{1}{100}\right)$ sans saisir un à un les éléments.

(b) Compléter la commande précédente pour qu'elle renvoie $\sum_{k=1}^{10} \frac{1}{k^2}$.

2. En une seule ligne de commande, calculer la somme $\sum_{n=0}^{10} \frac{1}{2^n}$.

3. Que calculent les commandes suivantes :

- `x = np.ones(10) ; y = np.cumsum(x)`
- `x = np.ones(10) ; y = np.sum(np.cumsum(x))`
- `x = np.ones(10) ; y = np.sum(np.cumsum(np.cumsum(x)))`

Exercice 5 (★★)

Le nombre de véhicules passant sur une petite route de campagne en une journée est une variable aléatoire X qui suit une loi de Poisson $\mathcal{P}(5)$.

On va simuler une année de circulation sur cette route. Pour cela, importons la bibliothèque `numpy.random` à l'aide de l'instruction `import numpy.random as rd`, et utilisons la commande `A = rd.poisson(5, [52, 7])`. La i -ème ligne de A correspond à la i -ème semaine de circulation. On obtient ainsi une matrice A qui contient 52×7 nombres tirés suivant la loi $\mathcal{P}(5)$.

- Déterminer le nombre maximal de véhicules circulant sur la route en une journée durant l'année. Et le nombre maximal de véhicules un mercredi durant l'année ?
- Déterminer les numéros des semaines où la route a connu son maximum de fréquentation.
- Calculer le nombre moyen de véhicules par jour durant ces 364 jours. Recommencez plusieurs fois en recréant la matrice A . Que constatez-vous ? Était-ce prévisible ?
- En une seule ligne de commande, évaluer la probabilité qu'une journée voit passer plus de 8 véhicules. Sauriez-vous calculer la valeur exacte de cette probabilité ?
- Toujours en une seule ligne de commande, déterminer le nombre de jours où la route a connu son minimum de fréquentation.

Exercice 6 (★★)

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 7 (★★★★)

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) :

$$\text{np.cumprod}(\text{np.arange}(n,0,-1)/\text{np.arange}(1,n+1))$$

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 $\llbracket 0, n-1 \rrbracket$.
 (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).
-

Exercice 8 (★)

Représenter sur une même figure les courbes des fonctions \sin , $x \mapsto x$, $x \mapsto x - \frac{x^3}{3!}$, $x \mapsto x - \frac{x^3}{3!} + \frac{x^5}{5!}$ sur l'intervalle $[-\pi, \pi]$. Que remarque-t-on ?

Exercice 9 (★)

Soient la fonction $f : x \in \mathbb{R} \mapsto x + 1 + e^{-x}$, et \mathcal{D} la droite d'équation $y = x + 1$.

Comme $\lim_{x \rightarrow +\infty} f(x) - (x + 1) = 0$, on dit que \mathcal{D} est *asymptote à la courbe \mathcal{C}_f de f au voisinage de $+\infty$* .

Illustrer graphiquement cette situation en écrivant des instructions qui permettent de tracer sur un même graphique \mathcal{C}_f et la droite \mathcal{D} . On fera varier x dans $[-2, 3]$.

Exercice 10 (★★ - Fonction réciproque)

1. On considère la fonction $f : \mathbb{R} \rightarrow \mathbb{R}$ définie par $f(x) = \frac{e^x - e^{-x}}{2}$ (appelée *sinus hyperbolique*, et souvent notée *sh*). Montrer que f réalise une bijection de \mathbb{R} dans \mathbb{R} .
 2. Définir une subdivision de l'intervalle $I = [-2, 2]$ en 100 sous-intervalles de même longueur, puis écrire des instructions permettant de tracer la courbe de f définie sur I .
 3. Ajouter des instructions pour tracer sur le même graphique la courbe de la bijection réciproque de f .
-

Exercice 11 (★★)

1. Montrer que la série $\sum \frac{(-1)^n}{n^2}$ est convergente.
 2. Créer un vecteur ligne `u`, tel que pour tout $1 \leq i \leq 30$, `u[i-1]` soit égal à $\frac{(-1)^i}{i^2}$.
 3. Créer un vecteur `v` tel que pour $1 \leq i \leq 30$, `v[i-1]` soit égal à $\sum_{k=1}^i \frac{(-1)^k}{k^2}$.

4. Illustrer la convergence de la série en représentant graphiquement les points M_i de coordonnées $\left(i, \sum_{k=1}^i \frac{(-1)^k}{k^2}\right)$ pour $i = 1, \dots, 30$. Donner une valeur approchée à 10^{-1} près de $\sum_{n=1}^{+\infty} \frac{(-1)^n}{n^2}$.

Exercice 12 (★★ - Étude d'une suite implicite)

On note, pour tout $n \in \mathbb{N}^*$, (E_n) l'équation $x^3 + \frac{1}{n}x^2 + x - 2 = 0$.

- Soit $n \in \mathbb{N}^*$. Étudier les variations sur \mathbb{R}_+ de la fonction $f_n : x \mapsto x^3 + \frac{1}{n}x^2 + x - 2$.
En déduire que l'équation (E_n) admet une unique solution u_n sur \mathbb{R}_+ , et que u_n appartient à $[0, 1]$.
- On considère le script suivant :

```

1 | def f(n,x):
2 |     return x**3+(1/n)*x**2+x-2
3 |
4 | def suite(n):
5 |     v = np.linspace(0,1,1000)
6 |     i = 0
7 |     while f(n,v[i])<0 :
8 |         i = i+1
9 |     return v[i-1]
```

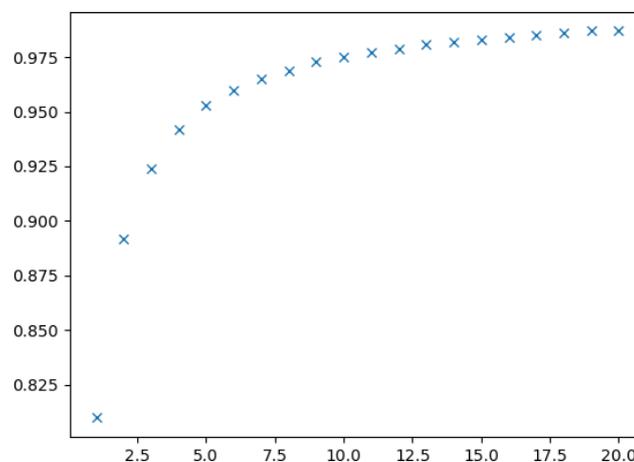
Que renvoie la commande `suite(n)` ?

- Compléter le programme suivant afin qu'il permette de représenter les 20 premiers termes de la suite $(u_n)_{n \in \mathbb{N}^*}$:

```

1 | U = np.zeros(20)
2 | for ...
3 |     U[k-1] = ...
4 | x = np.arange(1,21)
5 | plt.plot(...,...,'x')
```

- Le programme précédant complété permet d'obtenir la représentation graphique suivante :



Quelles conjectures peut-on émettre sur la monotonie et la limite de la suite $(u_n)_{n \in \mathbb{N}^*}$?

- Démontrer les résultats annoncés à la question précédente.