TP #

Formulaire Python

(Première année)

Importer une bibliothèque

Importations classiques : `import numpy as np`, `import matplotlib.pyplot as plt`, `import numpy.linalg as al`, `import numpy.random as rd`

On peut également importer "sans surnom" : `from numpy import *`, `from numpy.linalg import *`, etc... (moins recommandé par le programme)

Dans toute la suite, toutes les instructions débutant par `np.`, `plt.` `al.`, `rd.`, sont issues des bibliothèques correspondantes.
En pratique, on n'oubliera pas d'importer les bibliothèques nécessaires avant d'utiliser ces instructions !

Opérations basiques (avec la bibliothèque `numpy`)

Instruction	Description	Remarques	Exemple
`+ - * / **`	Addition, soustraction, multiplication, division, exposant pour des nombres réels (ou entiers).		`z = (x2-x1)**2 + (y2-y1)**2`
`np.e np.pi`	Valeur des constantes e et π.		`aire = np.pi * R ** 2`

`np.exp np.log`
`np.cos np.sin`
`np.sqrt np.abs`
`np.floor`
	Fonctions usuelles : exponentielle, logarithme néperien, cosinus, sinus, racine carrée, valeur absolue, partie entière. Peuvent être appliquées à des nombres réels (ou entiers).	Attention : pour ln, il faut bien utiliser `np.log` !
		L'instruction `np.ln` n'existe pas...
			`def gaussienne(x) :`
			`  y = np.exp(-(x**2)/2) / np.sqrt( 2 * np.pi );`
			`  return(y)`

`== < > <= >= !=` `and or not`	Tests de comparaison pour des nombres réels (ou entiers).	Ne pas confondre l'affectation `a=b` (a prend la valeur b) et le test `a == b` (est-ce que a est égal à b ?)	`if (a <= x) and (x <=b) :` `  print("x est entre a et b")`

Hors programme (mais utile)			

`%`	Modulo : si a et b sont des entiers, `a % b` renvoie le reste dans la division euclidienne de a par b.		`if (n % 2 == 0) :`
			`  print("n est pair !")`
			`else :`
			`  print("n est impair !")`

`x += a, x -= a`	Abréviation de `x = x + a` et `x = x - a`.		

Instructions d'affichage

Instruction	Description	Remarques	Exemple
`print`	• `print("mon texte")` affiche un texte, • `print(x)` affiche la valeur contenue dans la variable `x`.	On peut combiner ces deux types d'affichage en utilisant des virgules.	`x = 2;  y = 4*x + 1` `print("La variable x vaut", x, "et y vaut", y)`

Hors programme (mais utile)			
`x = input("texte")`	Affiche un texte, puis attend que l'utilisateur du programme entre une valeur. Cette valeur est ensuite conservée dans la variable `x`.		`n = input("Entrez une valeur entière")` `print("Le carré de", n, "est :", n**2)`

Listes

Instruction	Description	Remarques	Exemple
`L = [x1,x2,...,xn]`	Définition d'une liste "à la main".	La liste vide est `L = []`.	`MaListe = [2,-3.1,0.5,150]`
`L[i]`	Accède à l'élément d'indice `i` de la liste `L`.	Attention : le premier élément de la liste est `L[0]` !	`L = [1,2,3,7,5]` `L[3] = 4 # On modifie une valeur de la liste`

`range(n)`
`range(a,b)`
`range(a,b,r)`
	• `range(n)` crée la liste `[0,1,2,...]` qui s'arrête juste avant `n`. (si `n` est entier : `[0,1,2,...,n-1]`.)
	• `range(a,b)` crée la liste `[a,a+1,a+2,...]` qui s'arrête juste avant `b`. (si `a`,`b` sont entiers : `[a,a+1,a+2,...,b-1]`.)
	• `range(a,b,r)` crée la liste `[a,a+r,a+2r,...]` qui s'arrête juste avant `b`.
		Utile essentiellement pour définir des boucles `for`.	Ce programme affiche la somme des entiers de 1 à n :
			`S = 0`
			`for k in range(1,n+1) : # k = 1, 2, ..., n`
			`  S = S + k`
			`print(S)`

`L = [f(i) for i in ...]`	Crée une liste contenant les valeurs $f(i)$ pour i parcourant un ensemble de valeurs donné.	L'expression $f(i)$ peut être n'importe quoi mettant en jeu i et les opérations/fonctions usuelles !	Une manière rapide de créer la liste $[1, \frac{1}{2^2}, \frac{1}{3^2}, \ldots, \frac{1}{n^2}]$: `L = [1/i**2 for i in range(1,n+1)]`

Hors programme (mais utile)			

`len(L)`	Renvoie la longueur (len = "length") de la liste `L`, c'est à dire son nombre d'éléments.	Cette instruction fonctionne également pour des vecteurs (type `array`).	`n = len(L)`
			`for i in range(n) : # i = 0, 1, ..., n-1`
			`  print("Element d'indice", i, " : ", L[i])`

`L.append(x)`	Ajoute l'élément `x` à la fin de la liste `L`.		`Liste = [1,2,-3]` `Liste.append(12) # On ajoute 12 en fin de liste`
`del L[i]`	Supprime (del = "delete") la valeur d'indice `i` dans la liste `L`.		`L = [1,2,3,4]` `del L[2] # La liste devient : [1,2,4]`

Vecteurs : tableaux à une seule ligne (avec la biblothèque `numpy`)

Les tableaux à une seule ligne sont très similaires aux listes, à ceci près qu'il est possible d'effectuer des **opérations** sur les tableaux : addition, multiplication de tableaux, fonction appliquée à un tableau, etc... Pour cette raison, dans de nombreux cas, on préférera manipuler des vecteurs (type `array`) plutôt que des listes (type `list`).

Création de vecteurs

Instruction	Description	Remarques	Exemple
`X = np.array([x1,x2,...,xn])`	Définition d'un vecteur (tableau à une seule ligne) "à la main".		`X = np.array([1,-2.5,6,12])` `Y = np.array([i**3 for i in range(20)])`
`X[i]`	Accède à l'élément d'indice `i` d'un vecteur `X`.	Le premier élément est `X[0]` !	

`np.zeros(n)`	Crée un vecteur contenant `n` zéros.	On utilise souvent cette instruction pour créer un vecteur "de la bonne taille", que l'on peut ensuite remplir comme on le souhaite dans une boucle.	On dispose de la suite récurrente : $u_0 = 1$ et $\forall n \in \mathbb{N},\ u_{n+1} = 2u_n - 3$. Pour construire $X = [u_0, u_1, ..., u_{99}]$:
			`X = np.zeros(100); X[0] = 1`
			`for i in range(1,100) :`
			`  X[i] = 2 * X[i-1] - 3`

`np.ones(n)`	Crée un vecteur contenant `n` uns.		

`np.arange(n)`
`np.arange(a,b)`
`np.arange(a,b,r)`
	Fonctionne de la même façon que `range`, mais renvoie un vecteur (type `array`) plutôt qu'une liste.	C'est bien `arange` (le "a" de "array", suivi de "range"), et non pas `arrange`...	Pour contruire le vecteur `X`=[1,2,3,...,100] : `X = np.arange(1,101)`.

`np.linspace(a,b,n)`	Crée un vecteur contenant `n` nombres uniformément répartis entre `a` et `b`. (le premier élément est `a`, le dernier est `b`).	Souvent utile pour les représentations graphiques.	`X = np.linspace(0,1,100)` est un vecteur contenant 100 points répartis uniformément entre 0 et 1.

Opérations sur les vecteurs (coeff. par coeff)

Instruction	Description	Remarques	Exemple

`+ - * / **`	Opérations entre deux vecteurs de même taille (ou entre un vecteur et un nombre).	Ces opérations se font "coefficient par coefficient".	`X = np.array([1,-2,6])`
			`Y = np.array([1,0,2])`
			`Z = X + 2*Y` est le vecteur $[3, -2, 10]$.

`Y = f(X)`	Si `X` est le vecteur `[x1,x2,...,xn]` on construit ainsi le vecteur `Y = [f(x1),f(x2),...,f(xn)]`.	`f` peut faire appel aux "opérations basiques" ou bien aux fonctions usuelles.	Une manière rapide de créer le vecteur `Y`= $[1, \frac{1}{2^2}, \frac{1}{3^2}, \ldots, \frac{1}{n^2}]$:
			`X = np.arange(1,n+1)`
			`Y = 1/X**2`

`== < > <= >= !=`	Comparaison de deux vecteurs de même taille (ou d'un vecteur et d'un nombre).	Renvoie un vecteur de `True` et `False` donnant le résultat de la comparaison pour chaque coefficient.	Si `X` est le vecteur $[1, 0, 2]$ et `Y` est le vecteur $[3, -1, 1]$, alors `(X <= Y)` renvoie le vecteur `[True,False,False]`.

Représentations graphiques (avec la bibliothèque `matplotlib.pyplot`)

Instruction	Description	Remarques	Exemple

`plt.plot(X,Y)`	Si `X = [x1,x2,...,xn]` et `Y = [y1,y2,...,yn]` sont deux listes ou deux vecteurs de même taille, crée un graphe en reliant les points $M_1 = (x_1, y_1), M_2 = (x_2, y_2), \dots, M_n = (x_n, y_n)$ par des segments.	• Pour représenter une fonction f sur un intervale $[a, b]$, on choisira `X=np.linspace(a,b,100)` et `Y=f(X)`.
		• Pour représenter une suite, on construira par exemple `X`$= [0, 1, \dots, n]$ et `Y`$= [u_0, u_1, \dots, u_n]$.
			Création et affichage du graphe de $x \mapsto e^x$ sur le segment $[-1, 1]$:
			`X = np.linspace(-1,1,1000)`
			`Y = np.exp(X)`
			`plt.plot(X,Y)`
			`plt.show()`

`plt.hist(X)`	Si `X = [x1,x2,...,xn]` est une liste ou un vecteur, crée un histogramme à partir des données contenues dans `X`	• En ajoutant l'option `bins = ...`, on peut choisir en combien de "groupes" les données sont réparties (c'est à dire le nombre de "barres" dans l'histogramme)
		• En ajoutant l'option `density = True`, le diagramme est renormalisé pour que la somme des aires des bâtons soit égale à 1.
			Histogramme des résultats obtenus pour 100 lancers de dés :
			`X = rd.randint(1,6,100)`
			`plt.hist(X, bins = 6, density=True)`
			`plt.show()`

`plt.show()`	Affiche le ou les graphes/histogrammes construits.	On peut enchainer plusieurs `plt.plot` avant l'instruction `plt.show()` pour afficher plusieurs graphes sur une même figure.	

Hors programme (mais utile)			

Options d'affichage de `plt.plot`	On peut taper `plt.plot(X,Y,options)`, où `options` est une chaine de caractère qui détermine la couleur et le style du graphe.	• Couleur de la courbe :
		`b` (bleu), `g` (vert), `r` (rouge), `c` (cyan), `m` (magenta), `y` (jaune), `k` (noir), `w` (blanc).
		• Style des lignes :
		`-` `--` `:` `-.`
		• Style des points :
		`*` `o` `+` `x` `.` `d`
			Ligne continue rouge avec des étoiles à chaque point : `plt.plot(X,Y,?r-*?)`
			Uniquement des points représentés par des croix noires : `plt.plot(X,Y,'kx')`

Graphes légendés	Si on affiche plusieurs graphes sur une même figure, on peut ajouter l'option `label = ...` puis l'instruction `plt.legend()` avant le `plt.show()`.		`plt.plot (X, Y, ?b?, label = ?Courbe 1?)`
			`plt.plot (X, Z, ?r?, label = ?Courbe 2?)`
			`plt.legend()`
			`plt.show()`

Matrices : tableaux à plusieurs lignes (avec la bibliothèque `numpy`).

Création de matrices			
Instruction	Description	Remarques	Exemple
`A = np.array([[a11,a12,...,a1p], ... ,[an1,an2,...,anp]])`	Définition d'une matrice (tableau à plusieurs lignes) "à la main".	Attention à la syntaxe : il faut donner une "liste de listes" à l'intérieur des parenthèses !	`A = np.array([[1,0,-1],[1,2,5]])` Définit la matrice $A = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 2 & 5 \end{pmatrix}$.
`A[i,j]`	Accède à la valeur sur la i-ème ligne et j-ème colonne de la matrice `A`.	Attention : l'indexation des lignes et des colonnes démarre encore une fois à 0 !	Pour la matrice ci-dessus, `A[0,0]` vaut 1 et `A[1,2]` vaut 5.
`A[i,:]`	Vecteur contenant les valeurs présentes sur la i-ème ligne de la matrice `A`.	On peut ainsi "extraire" une ligne d'une matrice, ou bien modifier toute une ligne d'un coup.	Pour la matrice ci-dessus, après l'affectation `A[1,:] = [0,1,0]`, la matrice devient $A = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$.
`A[:,j]`	Vecteur contenant les valeurs présentes sur la j-ème colonne de la matrice `A`.	Même si on extrait une colonne, `A[:,j]` renvoie bien un vecteur ligne !	Pour la matrice juste au dessus, `A[:,1]` renvoie `array([0,1])`. Après l'affectation `A[:,2] = [5,5]`, la matrice devient $A = \begin{pmatrix} 1 & 0 & 5 \\ 0 & 1 & 5 \end{pmatrix}$.
`np.shape(A)`	Renvoie la taille `(n,p)` de la matrice `A`. Nombre de lignes : `np.shape(A)[0]`. Nombre de colonnes : `np.shape(A)[1]`	Le couple `(n,p)` est de type `"tuple"`, mais se manipule comme une liste.	Pour la matrice ci-dessus, `np.shape(A)` renvoie `(2,3)`.
`np.zeros((n,p))`	Crée une matrice de taille $n \times p$ contenant uniquement des zéros.	On peut aussi écrire `np.zeros([n,p])`.	
`np.ones((n,p))`	Crée une matrice de taille $n \times p$ contenant uniquement des uns.	On peut aussi écrire `np.ones([n,p])`.	
`np.eye(n)`	Crée la matrice identité de taille $n \times n$.		

Opérations sur les matrices (coeff. par coeff)			
Instruction	Description	Remarques	Exemple
`+ - * / **`	Opérations entre deux matrices de même taille (ou entre une matrice et un nombre).	`*` ne correspond **pas au produit matriciel** !	`M = -np.eye(3) + np.ones((3,3))`.
`B = f(A)`	Application d'une opération/fonction usuelle à chaque coefficient d'une matrice.		`B = np.exp(-A**2)`
`== < > <= >= !=`	Comparaison de deux matrices de même taille (ou d'une matrice et d'un nombre).	Renvoie une matrice de `True` et `False` donnant le résultat de la comparaison pour chaque coefficient.	Si $A = \begin{pmatrix} 2 & -1 \\ 1 & 3 \end{pmatrix}$ l'instruction `A > 1` renvoie : $\begin{pmatrix} \texttt{True} & \texttt{False} \\ \texttt{False} & \texttt{True} \end{pmatrix}$

Opérations à partir d'un échantillon de valeurs (avec la bibliothèque `numpy`)

Les opérations suivantes peuvent être réalisés sur une liste, un vecteur, ou une matrice.
Dans le cas d'une matrice, il est possible d'effectuer l'opération sur chacune des lignes (ou sur chacune des colonnes) et de renvoyer la liste des résultats ainsi obtenus.

Exemple :
- `np.sum(A)` renvoie la somme de tous les coefficients de la matrice `A`.
- `np.sum(A,0)` renvoie une liste contenant la somme calculée le long de chacune des colonnes de `A`.
- `np.sum(A,1)` renvoie une liste contenant la somme calculée le long de chacune des lignes de `A`.

Instruction	Description	Remarques	Exemple
`np.sum` `np.prod`	Calcule la somme/le produit des éléments d'une liste, d'un vecteur, d'une matrice.		Calcul rapide de $\sum_{k=1}^{n} \frac{1}{k^2}$: `np.sum([1/k**2 for k in range(1,n+1)])`.
`np.min` `np.max`	Calcule le minimum/maximum des éléments d'une liste, d'un vecteur, d'une matrice.		
`np.mean` `np.median` `np.var` `np.std`	Calcule la moyenne/médiane/variance/écart type des valeurs dans une liste, d'un vecteur, d'une matrice.	`std` = "standard deviation"	
`np.cumsum`	Renvoie une liste donnant la somme cumulative (suite des sommes partielles) d'une liste, d'un vecteur, d'une matrice.	Utile essentiellement pour calculer et représenter une fonction de répartition.	

Calcul matriciel, systèmes linéaires (avec les bibliothèques `numpy` et `numpy.linalg`)

Instruction	Description	Remarques	Exemple
`np.dot(A,B)`	Renvoie le produit matriciel AB.	Ne pas confondre avec `A*B` qui fait un produit "coeff par coeff" !	
`np.transpose(A)`	Renvoie la matrice transposée ${}^tA$.		
`al.matrix_power(A,n)`	Si `A` est une matrice carrée, renvoie la matrice A^n.	Ne pas confondre avec `A**n` qui calcule les puissances "coeff par coeff" !	
`al.matrix_rank(A)`	Renvoie le rang de la matrice `A`.		
`al.inv(A)`	Si `A` est une matrice carrée inversible, renvoie la matrice A^{-1}.		

`X = al.solve(A,Y)`	Si `A` est une matrice carrée inversible, renvoie l'unique solution X du système linéaire $AX = Y$.	• Cela reviendrait à calculer $X = A^{-1}Y$...
		• Les inconnues `X` et le second membre `Y` sont ici donnés en vecteur ligne !
			Pour résoudre $\begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$:
			`A = np.array([[1,2],[-1,1]])`
			`Y = np.array([1,3])`
			`X = al.solve(A,Y)`

Expériences et variables aléatoires (avec la bibliothèque `numpy.random`)

Les instructions suivantes génèrent des nombres aléatoires suivant les lois de probabilité usuelles.

A chaque fois, on fournit la valeur du ou des paramètres de la loi de probabilité (n et p pour une loi binomiale $\mathcal{B}(n,p)$, λ pour une loi de Poisson $\mathcal{P}(\lambda)$ etc...).

Par défaut, ces instructions renvoient une seule valeur. Si l'on veut plusieurs valeurs, on peut ajouter une option à la suite des paramètres :

- Un entier `m` pour générer un vecteur de taille m contenant des valeurs tirées indépendamment selon la loi de probabilité.
- Un couple d'entier `(m1,m2)` (ou `[m1,m2]`) pour générer une matrice de taille $m_1 \times m_2$ contenant des valeurs tirées indépendamment selon la loi de probabilité.

Instruction	Description	Remarques	Exemple

`rd.random()`
`rd.random(m)`
`rd.random((m1,m2))`
	Génère une ou plusieurs valeurs selon la loi $\mathcal{U}([0,1])$
	(nombre réel choisi "uniformément" dans le segment $[0,1]$).
		Si $p \in [0,1]$, il est utile de retenir que :
		`rd.random() < p` $= \begin{cases} \texttt{True} & \text{avec proba } p \\ \texttt{False} & \text{avec proba } 1-p. \end{cases}$
		De même pour le test `rd.random() <= p`.
			Lancer d'une pièce équilibrée :
			`if rd.random() < 1/2 :`
			`  print("Pile")`
			`else :`
			`  print("Face")`

`rd.randint(a,b+1)`
`rd.randint(a,b+1,m)`
`rd.randint(a,b+1,(m1,m2))`
	Génère une ou plusieurs valeurs selon la loi $\mathcal{U}(\llbracket a,b \rrbracket)$.	Attention : pour avoir un entier entre a et b, il faut donner les paramètres `a` et `b+1` !
		(même règle que pour `range`).
			Somme de 5 dés équilibrés :
			`X = rd.randint(1,7,5);`
			`print(np.sum(X))`

`rd.binomial(n,p)`
`rd.binomial(n,p,m)`
`rd.binomial(n,p,(m1,m2))`
	Génère une ou plusieurs valeurs selon la loi $\mathcal{B}(n,p)$.		Moyenne empirique pour 100 réalisations d'une loi $\mathcal{B}(6,\frac{1}{2})$:
			`X = rd.binomial(6,0.5,100)`
			`print(np.mean(X))`

`rd.geometric(p)`
`rd.geometric(p,m)`
`rd.geometric(p,(m1,m2))`
	Génère une ou plusieurs valeurs selon la loi $\mathcal{G}(p)$.		

`rd.poisson(lam)`
`rd.poisson(lam,m)`
`rd.poisson(lam,(m1,m2))`
	Génère une ou plusieurs valeurs selon la loi $\mathcal{P}(\lambda)$ (où $\lambda =$`lam`).		