L'essentiel à savoir en Python

Bibliothèques :

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rd
import pandas as pd
import numpy.linalg as al
```

1. Généralités

Afficher x	print(x)
Demander x	x = int(input('x =')) x=float(input('x ='))
Affecter y à x	x = y
Tester si x = y,x ≠ y, x ≥ y, x ≤ y	x==y, x!=y, x >=y, x<=y
Si … alors …	If condition : instruction
Si …alors … sinon …	If condition : instruction1 else : instruction2

Valeurs particulières : np.pi np.e

Listes

[0, …, n-1]	L=range(n)
$[n_1, …, n_2 – 1]$	L=range(n1, n2)
$[n_1, n_1 + h, n_1 + 2h, …, n_2 - h]$	L=range(n1,n2,h)
[a, …, b], n valeurs avec un pas constant	L=np.linspace(a,b,n)
Liste vide	L = **[]**
Liste définie par une expression	L = [expression for x in …] L = [x for x in … if …]

Opérations sur les listes

_ Ajouter x à la fin de L : **L.append(x)**

_ Supprimer L(k) : **del L[k]**

_ Concaténer deux listes : **L = L1+L2**

_ **L[i]** est le coefficient n°i de L (attention, l'indice du premier coefficient est 0 !).

_ **L[i:j]** est la liste extraite des éléments d'indice entre i et j-1.

_ tous les éléments : **L[:]**

_ longueur d'une liste L : **len(L)**

_ appartenance : **x in L** ⟶ True ou False

_ **L.count(x)** : nombre d'occurrences de x dans L

Boucles :

Si nombre d'itérations connu à l'avance	for i in range(n): instruction
Si nombre d'itérations inconnu	while condition : instruction

Suites, sommes et produits :

	Initialisation	Dans la boucle
Compteur	i = 0	i = i + 1 ou i+=1
Suite récurrente $\begin{cases} u_0 \in \mathbb{R} \\ u_{n+1} = f(u_n)\ \forall n \in \mathbb{N} \end{cases}$	u = u0	u = f(u)
Suites récurrentes d'ordre 2 $\begin{cases} u_0 \in \mathbb{R} \\ u_1 \in \mathbb{R} \\ u_{n+2} = f(u_n, u_{n+1})\ \forall n \in \mathbb{N} \end{cases}$	u = u0 v = u1	w = f(u,v) u = v v = w
Somme $\sum_{k=1}^{n} u_k$	s = 0	s = s + uk
Produit $\prod_{k=1}^{n} u_k$	p = 1	p = p * uk

Avec une fonction :

Si $x = [x_0, \dots, x_{n-1}]$,

np.sum(x) vaut $x_0 + \dots + x_{n-1}$

np.cumsum(x) vaut $[x_0, x_0 + x_1, \dots, x_0 + x_1 + \dots + x_{n-1}]$

(1^{ers} termes de la suite des sommes partielles)

np.prod(x) vaut $x_0 \times \dots \times x_{n-1}$

2. Fonctions

Définition d'une fonction :

```
def nomfonction(x) :
    y = …
    return y
```

Fonctions usuelles :

abs np.exp np.log np.sqrt

Dichotomie :

f continue et strictement monotone sur [a;b]. Soit $y_0 \in \mathbb{R}$. On admet que l'équation $f(x) = y_0$ admet une unique solution α sur [a;b]. Déterminer une valeur approchée de α à ε près.

```
a=...  ; b=….
while b – a > epsilon :
    c = (a + b)/2
    if f(c) > y0 :
        b=c      # si f est croissante (sinon échanger)
    else :
        a=c
print(a)    # ou b à la place de a
```

3. Courbes

Si x et y sont des listes de taille n :
plt.plot(x,y) trace la ligne brisée reliant les points $(x_i, y_i)_{1 \leq i \leq n}$.
plt.scatter(x,y) ou plt.plot(x,y,'*') tracent uniquement le nuage de points

plt.show() : Affiche le graphe

Courbe d'une fonction :

```
def f(x):          #définition de la fonction
  return …
x = np.linspace(a, b, n)   #subdivision  (ou x=np.arange(a, b+h, h))
y=[f(t) for t in x]    #images de la subdivision
plt.plot(x,y)
plt.show()
```

4. Matrices

Définir une matrice	M=np.array([[…, …],…, [… , ..]])
Matrice nulle	M = np.zeros((m,n))
Matrice ne contenant que des 1	M = np.ones((m,n))
Matrice identité	M = np.eye(n,n)
Matrice avec 1 sur une parallèle à la diagonale (0 ailleurs)	M = np.eye(n,n,i)
i-ème ligne de M	M[i]
j-ème colonne de M	M[:,j]
Opérations coefficient par coefficient	M* N, M / N, M ** n, …
Opérations sur les matrices	np.dot(A, B) al.inv(A) al.matrix_power(A, n)
Solution de AX = B	X = al.solve(A, B)
Transposée de M	np.transpose(M)
Rang de M	al.matrix_rank(M)
Valeurs propres et vecteurs propres de M	al.eig(A)

5. Probabilités

Si $p \in]0;1[$, **rd.random() < p** est un événement de **probabilité p**.

Loi	**avec rd.random()**	**directement**
$\mathcal{U}(\{n_1, \ldots, n_2\})$		x = rd.randint(n1,n2+1)

$\mathcal{B}(p)$	if rd.random()<p :
	  x = 1
	else :
	  x = 0
		x = rd.binomial(1,p)

$\mathcal{B}(n,p)$	x= 0
	for i in range(n) :
	  if rd.random()<p :
	  x = x + 1
		x = rd.binomial(n, p)

$\mathcal{G}(p)$	x = 1 while rd.random()>p : x = x + 1	x = rd.geometric(p)
$\mathcal{P}(\lambda)$		x = rd.poisson(lambda)
$\mathcal{U}([0;1])$	x=rd.random()	
$\mathcal{U}([a;b])$	x = a + (b – a)*rd.random()	
$\mathcal{E}(\lambda)$	x=-np.log(1-rd.random())/lamb	x=rd.exponential(1/lamb)
$\mathcal{N}(0,1)$		x = rd.normal(0,1)
$\mathcal{N}(m, s^2)$		x = rd.normal(m, s)

n valeurs indépendantes : rd.loi(paramètres, n)

Matrices de valeurs indépendantes : rd.loi(paramètres, [n,p])

6. Statistiques

Statistiques à une variable (x est un vecteur ou df un tableau de données) **:**

Moyenne empirique : $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$	np.mean(x) df.mean
Minimum, maximum	np.min(x), np.max(x)
Médiane	np.median(x) df.median()
Somme	np.sum(x)
Sommes "cumulées" (x_1, $x_1 + x_2$, …, $x_1 + \ldots + x_n$)	np.cumsum(x)
Variance empirique : $V(x) = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \overline{x})^2$	np.var(x) df.var()
Ecart-type empirique : $\sigma(x) = \sqrt{V(x)}$	np.std(x) df.std()
Etude générale de la série statistique	df.describe()
Par ligne ou colonne	np.mean(A,0) : moyenne par ligne np.mean(A,1) : moyenne par colonne idem pour min, max, sum, … df['NomColonne']

Diagramme en barres : Si x contient les valeurs prises et y les effectifs,
plt.bar(x, y) trace le diagramme en barres des modalités x avec les effectifs (ou fréquences) y.

Histogramme : si x est un vecteur, n un entier, et y un vecteur de réels dans l'ordre croissant
plt.hist(x,y) trace l'histogramme avec les classes définies par y.
plt.hist(x,n) trace l'histogramme avec n classes régulières entre le minimum et le maximum

Boite à moustache (ou boite de Tukey)
plt.boxplot(x)

Statistiques à deux variables (x et y sont deux vecteurs de même taille) :

Covariance : $\text{cov}(x, y) = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})$	np.cov(x,y)[0,1]

Méthode de Monte-Carlo
Soit $(X_n)_{n \geq 1}$ une suite de variables aléatoires indépendantes de même loi, d'espérance m, et qui admettent une même variance.
Alors $\overline{X_n} = \frac{X_1 + \ldots + X_n}{n}$ est un estimateur de m.

D'après la loi faible des grands nombres, $\overline{X_n}$ converge "en probabilité" vers m.

On peut utiliser **np.mean** pour calculer la moyenne.