⋆ Aide Mémoire - Python pour les Mathématiques ⋆
H2A - Lycée Stanislas

1 Algorithmes et fonctions

Les scripts doivent commencer par les lignes suivantes :

```
import matplotlib.pyplot as plt        # le module pyplot est importé et renommé plt
import numpy as np                     # le module numpy est importé et renommé np
import numpy.random as rd              # le module numpy.random est importé et renommé rd
```

Commande	Python	Remarques
Affectation et opérations arithmétiques : $+ \quad - \quad * \quad / \quad **$	`x = rd.randint(1,11)` `y = x ** 3`	On affecte à x une valeure entière aléatoire comprise entre 1 et 10 puis on la met au cube

Tests et logique : $== \quad > \quad < \quad >= \quad <= \quad !=$ `True` `False` `and` `or` `not` Structure conditionnelle simple	`x = rd.randint(1,11,3)`
	`if ((x[0] < x[1]) and (x[1] < x[2])) :`
	`  print("Ordre croissant")`
		x est un vecteur composé de trois entiers aléatoires compris entre 1 et 10

Structure conditionnelle avec elif et else	`x = rd.randint(1,11)`
	`if (x < 3):`
	`  x = x + 1`
	`elif(x > 7):`
	`  x = x - 1`
	`else:`
	`  x = 0`
		Si $x < 3$, alors on lui rajoute 1, sinon, si $x > 7$, alors on lui retranche 1, sinon on lui affecte la valeur 0

Boucle for avec range	`n = 100`
	`u = 0`
	`for k in range(1,n+1) :`
	`  u = np.sin(u)`
	`print("u :", u)`
		Calcul de u_n où $u_0 = 1$ et $\forall k \in \mathbb{N} \quad u_{k+1} = \sin(u_k)$ `range(1,n+1)` donne tous les entiers de 1 à n

Boucle for avec un vecteur Représentation graphique d'une suite	`n = 5`
	`T = range(0, n)`
	`A = np.zeros(n + 1)`
	`A[0] = 1`
	`for k in T :`
	`  A[k+1] = np.sin(A[k])`
	`x = np.linspace(0,n,n+1)`
	`plt.scatter(x, A, color = 'b', marker = '*')`
	`plt.show()`
		T contient n valeurs entières comprises entre 0 et $n-1$ Le tableau A contient les $n+1$ premiers termes de la suite `plt.scatter` permet de tracer des points

Boucle while	`n = 0`
	`u = 0`
	`while ( (2-u) > 0.0001 ) :`
	`  print("u",n," :", u)`
	`  n = n + 1`
	`  u = np.sqrt(u + 2)`
	`print("n :", n)`
		On étudie (u_n) où $u_0 = 0$ et $k \in \mathbb{N} \quad u_{k+1} = \sqrt{2+u_k}$ On cherche à déterminer à partir de quelle valeur de n, u_k donne une valeur approchée de $\lim\limits_{n\to+\infty} u_n$ à 10^{-4} près

Définition d'une fonction et tracé de sa courbe représentative avec titre et légende	`def  f(t) :`
	`  return  np.log(1+t**4)`
	`f = np.vectorize(f)  #on vectorise f`
	`x = np.linspace(1, 5, 100)`
	`y = f(x)`
	`plt.plot(x, y, 'b', linewidth = 1)`
	`plt.xlabel('x - axe des abscisses')`
	`plt.ylabel('y - axe des ordonnées')`
	`plt.title('Courbe représentative de f')`
	`plt.show()  #Affichage de la courbe`
		Définition de la fonction $f : t \mapsto \ln(1+t^4)$ On vectorise f pour pouvoir l'appliquer sur des vecteurs x contient 100 valeurs entre 1 et 5 b : blue, r : red, c : cyan, m : magenta, y : yellow, g : green, w : white

2 Calcul matriciel avec Numpy

Tous les scripts utilisant des tableaux doivent commencer par les lignes suivantes :

```
import numpy as np                # le module numpy est renommé np
```

Commande	Python	Remarques

Tableau à une ligne ou vecteur ligne	`L = np.arange(1,6)`
	`W = np.arange(0,9,3)`
	`U = np.zeros(5)`
	`V = np.linspace(1,7,4)`
		Création de $L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix}$
		(les éléments de L sont des entiers)
		Création de $W = \begin{pmatrix} 0 & 3 & 6 \end{pmatrix}$
		Crée un tableau à 5 cases ne contenant que des 0
		Création de $V = \begin{pmatrix} 1. & 3. & 5. & 7. \end{pmatrix}$
		(les éléments de V sont des réels)

Matrice quelconque	`B = np.array([[4,5,6],[7,8,9]])`	Création de $B = \begin{pmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

Matrices particulières
Initialisation
	`M = np.zeros((n,p))`
	`M = np.ones(3)`
	`M = np.ones((n,n))`
	`M = np.diag([1,2,3])`
		$M = (0)_{1 \leq i \leq n, 1 \leq j \leq p}$
		$M = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$
		$M = I_n$
		$M = diag(1,2,3)$

Opérations coefficient par coefficient : $+ \quad - \quad * \quad / \quad **$	`C = np.array([[0,1,0],[1,0,1]])` `M = C * B`	On obtient $M = \begin{pmatrix} 0 & 5 & 0 \\ 7 & 0 & 9 \end{pmatrix}$

Comparaison de deux matrices :
$== \quad > \quad < \quad >= \quad <= \quad !=$
	`if (np.all(B) == np.all(E)):`
	`  print("Matrices égales")`
	`else:`
	`  print("Matrices différentes")`
		La valeur renvoyée est un booléen
		qui vaut soit 0 (false) soit 1 (true)

Comparaison d'une matrices et d'un nombre :
$== \quad > \quad < \quad >= \quad <= \quad !=$
	`if (np.all(B) >= 0) :`
	`  print("Coeff. positifs")`
	`else:`
	`  print("Non tous positifs")`
		La valeur renvoyée est un booléen
		qui vaut soit 0 (false) soit 1 (true)

Taille de la matrice quelconque	`l, c = np.shape(B)`	`np.shape` renvoie deux coefficients ici on aurait `l = 2 et c = 3`
Transposition	`F = np.transpose(B)`	On obtient une matrice de $\mathcal{M}_{3,2}(\mathbb{R})$
Produit matriciel	`E = np.dot(F, B)`	On obtient une matrice de $\mathcal{M}_{3,3}(\mathbb{R})$
Longueur d'un vecteur ou d'une matrice	`Longueur = np.size(B)`	Renvoie la longueur du tableau B `Longueur` est donc égale à 6
Coefficient d'une matrice	`B[1,0]`	Coefficient de la matrice `B` situé à la **deuxième** ligne et à la **première** colonne égal à 7
Extraction d'une ligne Extraction d'une colonne	`L1 = B[0,:]` `C2 = B[:,1]`	Extraction de la première ligne de la matrice A, on obtient : $L1 = \begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$ Extraction de la deuxième colonne de B. : désigne toutes les lignes (ou toutes les colonnes)
Extraction	`H = B[0:2,1:3]`	Extraction d'une sous-matrice le résultat est : $H = \begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix}$
Concaténation horizontale Concaténation verticale	`C = np.zeros((2,2))` `K=np.concatenate((B,C),axis=1)`	On obtient $K = \begin{pmatrix} 4 & 5 & 6 & 0 & 0 \\ 7 & 8 & 9 & 0 & 0 \end{pmatrix}$
Nombres e et π	`x = np.e` `y = np.pi`	

Opérations usuelles sur les matrices ou les réels :	`G = np.exp(M)` `M = np.sqrt(E)`
	`H = np.log(B)` `N = np.abs(B)`
	`J = np.sin(B)` `P = np.floor(B)`
		On obtient par exemple $G = \begin{pmatrix} 1 & e^5 & 1 \\ e^7 & 1 & e^9 \end{pmatrix}$

Opérations en lien avec le cours de probabilités et de statistiques :
`np.sum` `np.min`
`np.max` `np.mean`
`np.cumsum` `np.median`
`np.var` `np.std`
	`S = np.sum(B)`
	`Q = np.sum(B, axis = 0)`
	`R = np.sum(B, axis = 1)`
		0 ajoute par colonnes, 1 ajoute par lignes
		si on ne marque rien, c'est sur toute la matrice
		On obtient $S = 39$
		$Q = \begin{pmatrix} 11 & 13 & 15 \end{pmatrix}$
		$R = \begin{pmatrix} 15 \\ 24 \end{pmatrix}$

3 Calcul matriciel avec Linalg et Random (Numpy)

Les scripts doivent commencer par les lignes suivantes :

```
import matplotlib.pyplot as plt          # le module pylot est importé et renommé plt
import numpy as np                       # le module numpy est importé et renommé np
import numpy.linalg as al                # le module numpy.linalg est importé et renommé al
import numpy.random as rd                # le module numpy.random est importé et renommé rd
import scipy.special as sp               # le module scpy.special est importé et renommé sp
```

Commande	Python	Remarques
Inverse	`A = np.array([[3,1],[1,3]])` `B = al.inv(A)`	Calcule l'inverse de $A = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$
Rang	`r = al.matrix_rank(A)`	Calcule le rang de A
Puissances	`n = 5` `A_n = al.matrix_power(A,n)`	Calcule A^5
Vecteurs propres Valeurs propres	`A = np.array([[3,1],[1,3]])` `D, P = al.eig(A)`	D est la matrice diagonale P est la matrice de passage
Résolution de système	`B = np.array([-1,2])` `X = al.solve(A, B)`	Résolution du système $AX = B$

Simulation d'une
loi géométrique,
histogramme
avec titre
et légende
	`p = 0.1  # paramètre p`
	`m = 1000  # nombre de simulations`
	`X = rd.geometric(p, m)`
	`plt.hist(X,bins = 31,range=(-0.5,30.5),density=True)`
	`plt.xlabel('Valeurs prises par la var')`
	`plt.ylabel('Fréquences')`
	`plt.title('Histogramme des valeurs prises par X')`
		bins : nbre. de classes
		range : valeurs extrêmes
		density = `True` :
		la somme des aires
		vaut 1 (histogramme
		normé)

Simulation d'une
loi binomiale
et histogramme
	`n = 20;  p = 0.7  # valeur de n et de p`
	`m = 1000  # nombre de simulations`
	`X = rd.binomial(n, p, m)`
	`plt.hist(X,bins = 21,range=(-0.5,20.5),density=True)`
		bins : nbre. de classes
		range : valeurs extrêmes
		`normed` = `True` :
		la somme des aires
		vaut 1

Simulation
d'un n-échantillon
d'une
loi de Poisson
	`lamb = 7  # paramètre lambda`
	`n = 10000  # nbre simulations`
	`X = rd.poisson(lamb, n)`	

Simulation
d'un n-échantillon
d'une loi
uniforme discrète
	`N1 = 1; N2 =11  # paramètres`
	`n = 10000  # nbre simulations`
	`X = rd.randint(N1, N2, n)`
		Loi uniforme
		discrète
		sur $[[N1, N2-1]]$

Simulation d'un n-échantillon d'une loi uniforme sur $[0,1]$	`n = 10000  # nbre simulations` `X = rd.random(n)`	

p simulations
de n var
indépendantes
suivant une
loi exponentielle
(tableau à p lignes
et n colonnes)
	`lamb = 0.1  # paramètre lambda`
	`esp = 1 / lamb  # paramètre lambda`
	`p = 1000  # nbre simulations`
	`n = 100  # nbre var`
	`X = rd.exponential(esp, (p,n)) # C'est l'espérance !`
		Utilisé dans le cours
		de statistiques pour
		simuler p fois
		un n-échantillon i.i.d.
		d'une loi et mettre
		en évidence une convergence
		en loi ou en probabilité

Simulation
d'un n-échantillon
d'une
loi normale
	`m = 5  # espérance`
	`v = 2  # variance`
	`n = 1000  # nbre simulations`
	`X = rd.normal(m, v, n)`
		Pour obtenir un tableau à p lignes
		et n colonnes contenant p simulations
		de n var indépendantes suivant
		la loi $\mathcal{N}(5,2)$, on remplace n par (p,n)

Simulation
d'un n-échantillon
d'une loi $\gamma(\nu)$
	`nu = 5  # paramètre nu`
	`n = 10000  # nbre simulations`
	`X = rd.gamma(nu, 1, n)`
		Pour avoir
		la loi $\gamma(\nu)$, on met le
		deuxième paramètre à 1

Fonction Φ	`x = 5  # valeur de x` `y = sp.ndtr(x)  # Valeur prise par Phi(x)`	Calcul de $y = \Phi(x)$

4 Fonctions de plusieurs variables

Les scripts doivent commencer par les lignes suivantes :

```
import matplotlib.pyplot as plt              # e module pylot est importé et renommé plt
import numpy as np                           # le module numpy est importé et renommé np
from mpl_toolkits.mplot3d import Axes3D      # le module Axes3D est importé
```

Commande	Python	Remarques

Définition d'une fonction de deux variables et vectorisation	`def f(x,y) :`
	`  return x**2 - y**2`
	`f = np.vectorize(f)`
		On définit la fonction $f : (x, y) \mapsto x^2 - y^2$ puis on la vectorise pour qu'elle puisse prendre en argument un tableau

Tracé du graphe d'une fonction de deux variables avec légende et titre	`X = np.arange(-1, 1, 0.02)`
	`Y = np.arange(-1, 1, 0.02)`
	`X, Y = np.meshgrid(X, Y)`
	`Z = f(X, Y)`
	`ax = Axes3D(plt.figure())`
	`ax.plot_surface(X, Y, Z)`
	`plt.xlabel('x - axe des x')`
	`plt.ylabel('y - axe des y')`
	`plt.title('Graphe de f')`
	`plt.show()`
		X et Y contiennent des valeurs comprises entre -1 et 1 par pas de 0.02
		`np.meshgrid` crée une matrice à partir des vecteurs X et Y
		f ayant été vectorisée, on peut calculer les différentes valeurs de Z en une seule étape, les deux commandes
		`ax = Axes3D(plt.figure())`
		et `ax.plot_surface(X, Y, Z)`
		permettent de tracer le graphe de f

Tracé des lignes de niveau	`plt.contour(X,Y,Z)` `plt.show()`	Permet de tracer les lignes de niveau du graphe de f
Tracé de gradients	`plt.quiver(X,Y,Z)` `plt.show()`	Permet de tracer les vecteurs gradient