MEMO Python pour ECG

Lycée KLEBER

On commence ce document par les modules qui sont nécéssaire à chaque programme, puis on développera les commandes nécessaire(exigible) pour le programme ECG approfondi en suivant le même ordre que le cours.

1 Les modules

Les instruction

```
from nom_du_module1 import *    #importe tous les éléments du module1
import module2 as mod2
from module3 import sous_module as smod
```

Pour la deuxième méthode, les fonctions du module2 devront être appelées par

```
mod2.fonction().
```

On l'utilise pour ne pas qu'il y ait d'écrasement d'autres fonctions, pour ne pas encombrer l'espace des noms. Pour les sous modules, on fera

```
smod.fonction().
```

Voici les principaux modules :

- `math` : pour importer les fonctions mathématiques usuelles et certaines constantes usuelles comme π (pi).
- `numpy` : pour utiliser le type array (tableau dont les éléments sont tous du même type, pratique pour les vecteurs, les matrices).
- `scipy` : outils nécessaires au calcul matriciel. Il contient un sous-module qui nous servira pour la partie aléatoire.
- `numpy.random` :le sous module, dédié aux simulations de variables aléatoires.
- `matplotlib` : pour générer des graphiques.

Le début de chaque script (programme) commencera donc de la façon suivante :

```
from math import *
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stat
```

On importe la bibliothèque `numpy.random` en écrivant l'une ou l'autre des instructions suivantes

```
from numpy.random import *
import numpy.random as rd
```

On utilisera la deuxième importation pour ce document.

2 Commandes de base

Action	scripte en python	Sortie
Types de variables - fonctions de conversions		
Type int, nombre entier	`N=760`	`760`
Converti si possible un décimal ou texte en entier	`int`(15.3)	`15`
Type float, nombre décimal	`X=3.76`	`3.76`
Convertit si possible un entier ou texte en décimal	`float("-11.24e8")`	`-1124000000.0`
Chaine de caractères , définie en la délimitant par des `" "`	`y="-11.8"`	`"-11.24e8"`
Convertit un nombre en chaîne	`str(3.76 )`	`'3.76'`
Type boolean Logique ne prend que deux valeurs : `True` et `False`	`b=(2*3==6)`	`True`
Liste : `list`	`L =[1,'a',-13],L[0]`	`[1,'a',-13],1`
Ensemble : `set`	`S={18,'a',3},S[1]`	`{1,2,3},'a'`
p-uplet : `tuple`	`T=(-4,'a',17), T[2]`	`(-4,'a',17),17`
Tableaux ou matrice : `array`	`M=np.array([[1,2,3],[4,5,6]])`	`array([[1, 2, 3], [4, 5, 6]])`
Affectation, implémentation, ...		
Assigner ou affecter	`x = a`	Stocke la valeur de `a` en `x`
Assignation (multiple) permutation	`a, b = 0, 1`	Pareil que `a = 0` et `b = 1`
Incrémenter ou implémenter	`x+ = a` ou bien `x = x + a`	Affecte à `x` la valeur de `x + a`
Décrémenter	`x− = a` ou bien $x = x - a$	Affecte à `x` la valeur de `x − a`
Afficher (sortie écran)	`print`("`Bonjour`")	Affiche seulement le mot : `Bonjour`
Entrée donner par l'utilisateur	`age = input`("`Quel age as tu`?")	On affecte à la variable `age` la valeur donner par l'utilisateur
Type d'une variable	`type`(`variable`)	Renvoie le type de la variable
Catalogue des variables	`dir`(`variable`)	Renvoie les fonctions, les listes ...
Commentaires	#	Le reste de la ligne après # est un commentaire
Aide en ligne	`help`(`commande`)	Renvoie à l'aide

Remarque : La conversion d'un objet en un autre type se fait de la façon suivante :

```
nouveau=nouveautype(ancien)
```

3 Opérations usuelles et opérateurs logiques.

Action	scripte en python	Sortie
Opérations classiques		
Addition : +	`x = 2 + 5`	`x = 7`
Multiplication :*	`y = 3 * 6`	`y = 18`
Puissance : **	`z = 4 * *2`	$z = 16$
Division décimale : /	`t = 27/2`	`t = 13.5`
Quotient de la division euclidienne : //	`q = 27//2`	`q = 13` : on affecte à `q` le quotient de la division euclidienne de `27` par `2`
Le reste de la division euclidienne : %	`r = 27%2`	`r = 1` : on affecte à `r`le reste de la division euclidienne de `27` par `2`
Opérateurs logiques et de comparaison		
supérieur	`x > y` ou `x >= y`	Renvoie `True` si $x > y$ ou `x = y` `False` sinon
inférieur	`x < y` ou `x >= y`	Renvoie `True` si $x < y$ ou `x = y` `False` sinon
Egalité	$x == y$	Renvoie `True` si `x = y` `False` sinon
Différent	`x!=y`	Renvoie `True` si `x` $\neq$ `y` `False` sinon
L'opérateur disjonction 'ou'	`P or Q`	Renvoie `True` si et seulement si l'un au moins est vraie ;
L'opérateur conjonction 'et'	`P and Q`	Renvoie `True` si et seulement si les deux propositions sont vraie ;
Opérateur négation	`not P`	Renvoie `True` si `P` est faux Renvoie `False` si `P` est vraie
Opérateur appartient	`a in Li`	Renvoie `True` si `a` appartient à la liste `Li`. `False` si `a` appartient à la liste `Li`
Entrées, sorties console, opérations numériques		
Entrée	`input()`	lit un texte saisi au clavier . Renvoie toujours une chaîne de caractères.
Conversion possible en nombre entier par `int( )`	`n=int(input('n=') )`	affiche n= dans la console et affecte l'entier donnée par l'utilisateur à la variable n
Conversion possible en nombre entier par `float( )`	`x=float(input('x=') )`	affiche x= dans la console et affecte le réel donnée par l'utilisateur à la variable x
Remarque : On peut aussi utiliser la fonction `eval()`pour les conversion : `x=eval(input('x=') )`		
Sortie en console	`print( 'La valeur est ',x)`	affiche en console : la valeurs est puis le contenu de la variable x en les séparant par une tabulation.
Il est possible d'avoir des sortie graphique à l'aide des fonctions `plot, bar` et `hist` qu'on verra dans la partie graphique de ce document.		

4 Généralité des listes/tableaux

Action	Scripte en python	Sortie
Longueur d'une liste	`len([1,2,'a'])`	3
Ajouter un élément à la fin de la liste. On affecte à `L` la liste `[1,2,4,'a']`	`L.append(a)`	Aout de l'objet `a` en fin de liste L
Rajouter à une liste un élément	`L.append(4)`	Rajoute 4 la fin de la liste et renvoie `[1,2,'a',4]`
Concaténer deux listes	`L+[-17]`	`[1,2,'a',4,-17]`
Répéter une liste 2 fois	`2*[-3,'b']`	`[-3,'a',-3,'a',-3,'a']`
Expulser un élément	`L.pop()`	Expulse le dernier élément de L
Trier	`L.sort()`	Trie L par ordre croissant
Symétrie centrale	`L.reverse()`	Inverse les éléments de la liste
Tester l'appartenance	`1 in [1,2,3]` ou `5 in [1,2,3]`	`True` ou `False`
Extraction de la tranche	`L[i:j]`	`[L[i], ... , L[j-1]]`
Extraction de la tranche avec un pas p	`L[i:j:p]`	De même de p en p à partir de `L[i]`, tant que `i+k*p < j`
Format tableau	`L.shape`	renvoie un tuple qui contient (nbre lignes, nbre colonnes,...)
Extraction :`Liste=[0,1,2,3,4,5]`	`Liste[0:3]`	`[0,1,2]`
Extraire tableau 2d `Liste=array([[1,2,3], [4,5,6] ])`	`a[0:2,0:2]`	`[[1,2],[4,5]]`
Compter l'apparition d'une occurence `Liste=[-3,-3,17,14]`	`Liste.count(-3)`	2 : le nombre d'occurence (-3)
Maximum d'une liste `max(liste)`	`max([-3,-3,17,14])`	17
Minimum d'une liste `min(liste)`	`min([-3,-3,17,14])`	-3
Créer une copie de L	`L1=L.copy()`	Crée un nouveau pointeur L1
L'indice d'un élément dans une liste	`L.index(a)`	Position de la première occurrence de a
Insertion d'un élément a dans la liste L	`L.insert(i,a)`	Insertion de l'objet a en position i
Générer liste d'entiers	`range(6)`	Renvoie la liste [0,1,3,4,5]
`range(a,b)` $\rightarrow [\![a,b[\![$	`range(6,9)`	Renvoie [6,7,8],non pas le dernier
Exemple	`linspace(0,10,5)`	`array([0.,2.5,5.,7.5,10.])`
Liste définie en compréhension		
`[expr for element in iterateur if condition]`	Liste formée des valeurs `expr` quand `element` parcourt `iterateur` Optionnel : `if condition`. Seuls les éléments vérifiant la condition sont insérés dans la liste.	
`[i for i in range(10) if i%3==0]`	`Python renvoie -->`	`[0,3,6,9]`
`[expr for el1 in it1 for el2 in it2...]`	Où l'on peut mettre plusieurs itérateurs	
`[i**j for i in range(1,4) for j in range(1,3)]`	`Python renvoie -->`	`[1,1,2,4,3,9]`
Conversion L $\rightarrow$ tableau	`reshape(liste,(3,2))`	L a 6 éléments $\rightarrow$ tableau 3 x 2

5 Matrice ou tableau en appelant le module Numpy

En important le module Numpy de la façon suivant `import numpy as np` on peut écrire les instructions du tableau ci-dessous. Ne pas oublier de précéder les instruction par `np.`

Action	Scripte en python	Sortie
Générer subdivision de $[a, b]$	`np.linspace(a,b,n)`	Renvoie un vecteur ligne de n valeurs régulièrement espacées entre a et b : l'espace entre les valeurs
np.array(L) Unidimensionnel	`V=np.array([1,2,-1])`	Pour construire un array, où L est une liste. V est vecteur ligne à 3 composantes
Bidimensionnel	`A=np.array([[1,2-1],[2,3,-6]])`	Matrice 2 lignes, 3 colonnes
np.array(L,int), np.array(L,float)	`np.array([1,2,-1.5],int)`	Pour imposer le type des éléments de l'array et renvoie `array([ 1, 2, -1])`
Dimension d'une matrice T à l'aide de la fonction `len`	`len(T)`	Nombre d'éléments de T si unidimensionnel, nombre de ligne si bidimensionnel.
Dimension d'une matrice T à l'aide de la fonction `shape`	`shape(T)`	Renvoie le format de A sous forme d'un tuple `(nb_ligne,nb_colone)`.
Copier une matrice	`np.copy(T)`	Réalise une copie de T

Extraction	`V[i]`
	
	`A[i,j]`
		Elément de V en position i.
		Elément de A en position (i,j).
		Attention le premier élément a pour indice 0.

`A[i1:i2,j1:j2]`	`A[2:4,3:5]`	Renvoie les éléments de A compris entre les indices de ligne i1 et i2-1 et indices de colonnes j1 et j2-1.
Autres règles `[i:j:p]` pour indices par pas de p, i : tous les indices à partir de i, :j tous les indices jusqu'à j		
Matrices et vecteurs prédéfini		
Vecteur contenant les entiers de m à $n-1$ `np.arange(m,n)`	`np.arange(5,10)`	`array([5, 6, 7, 8, 9])`
vecteur nulle à n composantes : `np.zeros(n)`	`np.zeros(4)`	`array([0,0,0,0])`
Matrice nulle d'ordre $n \times p$ `np.zeros((n,p))`	`np.zeros((2,4))`	`array([[0., 0., 0., 0.],` `[0., 0., 0., 0.])`
Vecteur de n,composantes égaux à 1 : `np.ones(n)`	`np.ones(3)`	`array([1., 1., 1.])`
ou matrice d'ordre $n \times p$ `np.ones((n,p))`	`np.ones((2,3))`	`array([1., 1., 1.],` `[1., 1., 1.])`
Matrice identité d'ordre $n \times n$ np.eye(n)	`np.eye(2)`	`array([1., 0.],` `[0. 1.])`
Matrice diagonale dont la diagonale est V : `np.diag(V)`	`np.diag(np.array([1,2]))`	`array([1., 0.],` `[0. 2.])`

6 Opérations sur les matrices

Pour tester les opérations matricielles nous choisirons les matrices suivantes :

$$A = \begin{pmatrix} 9 & -3 \\ -4 & 5 \end{pmatrix}, B = \begin{pmatrix} 2 & 4 \\ 6 & -8 \end{pmatrix} \text{ et } X = \begin{pmatrix} 9 \\ 5 \end{pmatrix}$$

Nous écrirons les matrices A et B, sur Python de la façon suivant :

```
import numpy as np
A= np.array([[9, -3],[-4, 5]]);B=np.array([[2, 8],[6, -8]]);X=np.array([[9],[ 5]])
```

Action	Scripte en python	Sortie
Additionner terme à terme, les éléments d'une matrice(tableau)	`A+B`	`array([[11, 5], [ 2, -3]])`
Produit matriciels, lorsque cela est possible	`np.dot(A,B)`	`array([[0, 96], [ 22, -72]])`
Le produit `A.X` renvoie une matrice colonne $\begin{pmatrix} 9 \\ 5 \end{pmatrix}$	`np.dot(A,X)`	`array([[ 66], [-11]])`
Produit terme à terme des élément	`A*B`	`array([[18, -24], [-24, -40]])`
Multiplier les coefficient de la matrice `A` par un réel a : `a*A`	`a*B`	`array([[ 6, 24], [ 18, -24]])`
Pour concaténer deux matrices, par défaut verticalement (l'une en dessous l'autre)	`np.concatenate((A,B))`	`array([[ 9, -3], [-4, 5], [ 2, 8], [ 6, -8]])`
Pour concaténer deux matrices, horizontalement(l'une à côté l'autre)	`np.concatenate((A,B),axis=1)`	`array([[ 9, -3, 2, 8], [-4, 5, 6, -8]])`
Applique la fonction `cos,sin,ln exp` ... à tous les éléments d'une matrice A (et autres fonctions)	`np.exp(A)`	array([[8.10308e+03, 4.97e-02], [1.8315e-02, 1.4841e+02]])
Inverse la matrice A	`np.linalg.inv(A)`	array([[0.15151515, 0.09090909], [0.12121212, 0.27272727]])
Résout le système$AY = X$	`np.linalg.solve(A,X)`	`array([[1.81818182], [2.45454545]])`
Déterminant d'une matrice A	`np.linalg.det(A)`	33.000000000000014
Rang d'une matrice A	`np.linalg.matrix_rank(A)`	2
Trace d'une matrice	`np.trace(B)`	-6
Transposée d'une matrice	`A.transpose()`	array([[9, -4], [-3, 5]])
Puissance 3ème de A	`linalg.matrix power(A,3)`	array([[1005, -489], [-652, 353]])
Les valeurs propres d'une matrice A, retourné sous forme d'un vecteurs.	`np.linalg.eigvals(A)`	`array([11., 3.])`
Sortie de vecteur formé des valeurs propre et la matrice constituée de valeurs propre associés dans l'ordre de ce dernier(matrice de passage)	`np.linalg.eig(A)`	(array([11., 3.]), array([[0.83205029, 0.4472136], [-0.5547002 , 0.89442719]]))

Commandes générales fonctions

Action	Scripte en python	Sortie
Fonctions usuelles	`log, exp,cos, sin,`	*log* : logarithme népérien, exp : fonction exponentielle, ...
La constante π	`pi`	3.141592653589793
La partie entière inférieur	`floor(5.7)`	
La partie entière supérieur	`ceil(5.2)`	6
La valeur de *e*donnée par :	`e`	2.718281828459045
valeur absolue	`abs(-3)`	3
`sqrt`	`sqrt(16)`	4
Bloc de condition		

Instruction conditionnel1
`if condition :`
`  conclusion`
	`if x==0:`
	`  x=x+1`
		Si `x` est égal à `0` alors on
		rajoute1 à `x`
		Fin de l'instruction

Instruction conditionnel2
`if condition :`
`  conclusion 1`
`else :`
`  conclusion 2`
	`if x==0:`
	`  x=x+1`
	`else :`
	`  x=x-1`
		Si x est égal à 0 (test)
		alors implémente x de 1
		sinon
		décrémente x de 1
		Fin de condition

Instruction conditionnel 3
`if condition 1 :`
`  conclusion 1`
`elif condition 2 :`
`  conclusion 2`
`else :`
`  conclusion 3`
	`if x%3==0:`
	`  r=0`
	`elif x%3==1:`
	`  r=1`
	`else :`
	`  r=2`
		Si le reste de la division
		euclidienne est 0, alors `r=0`
		sinon Si le reste de la division
		euclidienne est 1, alors `r=1`
		sinon `r=2`
		Attention à l'indentation et les :
		la fin de la ligne de if, elif et else

Boucles for et while		

Boucle for lorsqu'on connait les
bornes (le nombre des itérations)
`for i in ....:`
`  instructions`
⋮
	`u=0.5`
	`for i in range(n+1):`
	`  u=0.5*u*(u-3)`
	`print(n) #ici on est en`
	`#dehors de la boucle for`
		La boucle calcul les n terme de
		la suite (u_n) définie par $u_0 = 0,5$
		et $u_{n+1} = 0,5u_n(u_n - 3)$,
		affiche u_n, **Attention** à
		l'indentation et les deux
		à la fin de la ligne de `for`

Boucle while lorsqu'on a une
condition réalisable, pour
que la boucle s'arrête
`While condition :`
`  instructions`
⋮
	`u=0.5 ; n=1`
	`while u<2.32:`
	`  u=0.5*u*(u-3)`
	`  n=n+1`
	`print(n)`
	`# print(n)est en dehors de`
	`la boucle for`
		Cette boucle calcul le nombre
		d'itérations n nécessaire, pour
		que u_n soit supérieur à 2.32,
		puis affiche n.
		Attention à l'indentation et
		les deux points à la fin de la ligne
		de `while`

Déclaration d'une fonction :
`def nomfct(arg1,arg2,...):`
`  Instruction 1`
⋮
`  return sort1,sort2,...`
`#ici on est en dehors`
`# de la fonction`
	`def suite(n,a):`
	`  u=0.5`
	`  for i in range(n+1):`
	`  u=a*u*(u-3)`
	`  return u`
		La fonction `suite` calcule et
		affiche le n^{ème} terme la suite
		$(u_n)_n$ sachant que a et n
		sont données par l'utilisateur.
		`suite` est le nom de
		la fonction. $a$et n sont les
		arguments de la fonction `suite`

7 Probabilité : simulation de variables aléatoires

Nous avons besoin, pour les simulations de variables aléatoires, à importer les bibliothèques et libraires nécessaires à chaque situation :

`numpy()` et `numpy.random` de la façon suivante :

```
from numpy.random import *
import numpy.random as rd
```

Action	Scripte en python	Sortie
Simule la loi uniforme continue $\mathscr{U}([0,1[)$	`rd.random()`	0.0733833
Simule r réalisations de la loi $\mathscr{U}([0,1[)$	`rd.random(3)`	array([0.664, 0.996, 0.0422])
`rd.random([r,s])`simule `r`×`s` réalisations de la loi $\mathscr{U}([0,1[)$ sous la forme d'une **matrice de**$\mathscr{M}_{r,s}(\mathbb{R})$.	`rd.random([r,s])`	array([[0.143, 0.031, 0.027], [0.941, 0.942, 0.307]])
L'instruction `rd.random()<=p` envoie un booléen qui prend la valeur True ou False	`rd.random()<=0.3`	True
L'instruction `rd.random(r)<=p` envoie le vecteur u de r composante booléen, qui prennent la valeur True ou False	`u=rd.random(3)<=0.3`	`u=array([False,False,False])`
Le nombre de booléens qui ont pris la valeur `True` (True =1 et False=0).	`np.sum(u)`	0
Calcul de la moyenne des booléens (ou proportion) qui ont pris la valeur `True` dans u	`np.mean(u)`	0
`rd.randint(n)`simule la loi uniforme sur $[\![0, n-1]\!]$ avec $n \in \mathbb{N}$.	`rd.randint(3)`	1
`rd.randint(a,b)`simule la loi uniforme sur $[\![a, b-1]\!]$ avec $a < b$.	`rd.randint(3,9)`	4
`rd.randint(a,b,c)`simule la loi uniforme sur $[\![a, b-1]\!]$ avec $a < b$ et renvoie un vecteur de c composantes.	`v=rd.randint(1,9,3)`	`v=array([8, 7, 5])`
`rd.randint(a,b,[m,n])`simule la loi uniforme sur $[\![a, b-1]\!]$ avec $a < b$ et renvoie une matrice d'ordre $m \times n$.	`M=rd.randint(4,16,[2,3])`	`M=array([[12, 4, 13], [ 8, 4, 6]])`
`rd.binomial(n,p)` simule la loi binomiale $\mathcal{B}(n,p)$ renvoie le nombre de succès réalisés au bout de n lancers.	`rd.binomial(10,0.2)`	3
`rd.binomial(n,p,nb_exper)` renvoie un vecteur `de nb_exper` composantes . Chaque composante suit la loi binomiale $\mathcal{B}(n,p)$.	`rd.binomial(10,0.2,5)`	`array([2,2,3,1,2])`
`rd.binomial(n,p,[r,s])` renvoie une matrice d'ordre `r`×`s`	`rd.binomial(10,0.2,[2,3])`	`array([[1, 2, 0], [1, 0, 2]])`
`rd.geometric(p)` simule la loi géométrique $\mathcal{G}(p)$.	`rd.geometric(0.2)`	12

8 Probabilité : simulation de variables aléatoires (suite)

Action	Scripte en python	Sortie
`rd.geometric(p,n)` renvoie un vecteur `de n` composantes . Chaque composante suit la loi géométrique $\mathcal{G}(p)$.	`rd.geometric(0.2,4)`	`array([3,3,2,3])`
`rd.geometric(p,n,[r,s])` renvoie une matrice d'ordre `r`×`s`	`rd.geometric(0.2,4,[2,3])`	`array([6, 27, 4], [ 9, 2, 1]])`
`rd.poisson(lambda)`simule la loi de Poisson $\mathcal{P}(\lambda)$.	`rd.poisson(5)`	`12`
`rd.poisson(lambda,n)` renvoie un vecteur `de n` composantes . Chaque composante suit la loi de Poisson $\mathcal{P}(\lambda)$.	`rd.poisson(5,3)`	`array([5,7,2])`
`rd.poisson(lambda,[r,s])` renvoie une matrice d'ordre `r`×`s`	`poisson(5,[2,3])`	`array(4, 8, 8], [ 9, 1, 6]])`
`(b-a)*rd.random()+a` simule la loi uniforme continue $\mathscr{U}([a,b[)$	`3*rd.random()+1`	renvoie 2.5046708 suivant la loi $\mathscr{U}([1,4[)$
`rd.uniform(a,b)`simule aussi la loi uniforme continue $\mathscr{U}([a,b[)$	`rd.uniform(1,4)`	2.7676609
`rd.exponential(1/a)` simule la loi exponentielle $\mathcal{E}(a)$ de paramètre $a>0$	`rd.exponential(0.5)`	`0.065381763497`
`rd.exponential(1/a,n)` renvoie un vecteur `de n` composantes . Chaque composante suit la loi exponentielle $\mathcal{E}(a)$	`rd.exponential(0.5,3)` renvoi un réel suivant la loi $\mathcal{E}(2)$	`array([0.29, 0.045, 0.065])`
`rd.exponential(1/a,[r,s])` renvoie une matrice d'ordre `r`×`s`	`rd.exponential(0.5,[2,3])`	`array([[0.083, 0.700], [0.935, 0.287]])`
`rd.gamma(v)` simule la loi gamma $\gamma(v)$ de paramètre $v>0$	`rd.gamma(2)`	`6.1792773164424`
`rd.gamma(v,vecteur)` renvoie un vecteuur `de n` composante Chaque composante suit la loi gamma : $\gamma(v)$	`rd.gamma(2,[2,3,4])`	`array([5.43,3.83,2.34]))`
`rd.normal(m,sigma)` simule la loi normale $\mathcal{N}(m,\sigma^2)$ de paramètres $m\in\mathbb{R}$ et $\sigma>0$.	`rd.normal(5,0.1)`	4.984441772221091
`rd.normal(m,sigma,`[r,s]) renvoie $r\times s$ simulations de la loi normale de paramètres$m$ et σ^2.	`rd.normal(5,0.1,[2,3])`	array([[4.89029953, 4.93447703], [5.13218965, 4.91476433]])
`rd.normal()` simule la loi normale centrée réduite.	`rd.normal()`	1.1143096924632871
`rd.permutationl()` simule la loi normale centrée réduite.	`rd.permutationl(1,2,3,4])`	array([1, 4, 2, 3])

9 Représentation graphiques

Nous avons besoin dans cette partie de la bibliothèque `matplotlib.pyplot` que l'on importera ainsi :

```
import matplotlib.pyplot as plt
```

Lorsqu'on souhaite représenter graphiquement une fonction ou une suite à la main, il nous faut un tableau de valeurs ; autrement dit, les valeurs de $f(x)$ (ou u_n) pour un certain nombre de valeurs de x (ou de n).

9.1 Représentations graphiques en dimension deux

pour $a, b \in \mathbb{R}$ et $n \in \mathbb{N}$,la commande `np.linspace(a,b,n)` crée un tableau de n valeurs équiréparties de a à b inclus.

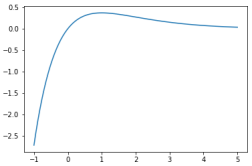
pour $a, b, p \in \mathbb{R}$, la commande `np.arange(a,b,p)` crée un tableau de valeurs de a inclus à b exclu avec un pas de p. Voici la structure ainsi que la syntaxe pour obtenir la courbe d'une fonction :

```
x = liste des abscisses
y = liste des ordonnées
plt.plot(x,y)
plt.show()
#Les listes des abscisses et des ordonnées peuvent être soit du type numpy.ndarray, soit du t
```

Exemple 1

Programme pour représenter la fonction $f : x \longmapsto xe^{-x^2}$ sur l'intervalle $[-1, 5]$

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-1,5,100)
y=x*np.exp(-x)
plt.plot(x,y)
plt.show()
```



Les commandes qui suivent ne sont pas exigibles à l'écriture, mais elles peuvent parfois servir :

- plt.grid() : fait apparaître une grille sur le fond du repère
- plt.axis('equal') : rend le repère orthonormé
- plt.axis([a,b,c,d]) : restreint le repère entre les abscisses a et b et les ordonnées c et d
- plt.plot(x,y,label="nom de la courbe")
- plt.plot(x,y,'couleur'), où couleur désigne la couleur voulue (ou son initiale)
- plt.legend() : affiche la légende

Exemple 2

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-1,5,100)
y=x*np.exp(-x**2)
plt.plot(x,y, label="Courbe de f")
x=np. linspace (0.01 ,5 ,100)
y=x*np.log(x)-x+1
plt.plot(x,y, label="Courbe de g")
```

```
plt.title("Courbes de f et g")
plt.legend()
plt .show()
```

9.2 Représentations graphiques de suites

Pour représenter une suites, nous considérons que les abscisses des points sont des entiers naturels et ces points serons représentés par des symboles + ou *... par exemple.

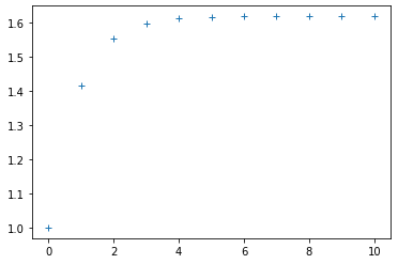
```
plt.plot(x,y,'+') # marque les points avec des +
plt.plot(x,y,'o') #marque les points avec des o
```

On souhaite représenter les termes de la suite (u_n) définie par :

$$\begin{cases} u_0 = 1 \\ \forall n \in \mathbb{N}, u_{n+1} = \sqrt{1+u_n} \end{cases}$$

On commencer par créer une fonction permettant les calculs des termes de $(u_n)_n$ puis créer une liste d'abscisses et une liste d'ordonnées :

```
import numpy as np
import matplotlib . pypl
def u(n):
    U=1
    for k in range(1,n+1
        U=np.sqrt(1+U)
   return U
x=range(0,11)
y=[u(n) for n in range (0 ,11)]
plt.plot(x,y,'+')
plt.show()
```



9.3 Diagramme en bâtons des probabilités théoriques et histogramme

9.3.1 Définition.

Si x et y sont des vecteurs de même taille, `plt.bar(x,y)` trace le diagramme en bâtons d'abscisse x et d'ordonnée y.

9.3.2 Définition.

Si x est un vecteur contenant une série statistique et c un vecteur contenant les classes choisies, la commande `plt.hist(x,c)` dessine l'histogramme associé à la série statistique x triée selon les classes définies par c.

9.3.3 Méthode. Comment tracer le diagramme en bâtons des fréquences ?

Pour tracer le diagramme des fréquences d'un échantillon x (qu'on suppose à valeurs entières), on procède ainsi :
(i) on décide des modalités $m_1 < m_2 < \cdots < m_k$ qu'on souhaite représenter ;
(ii) on définit les classes $c = (m_1 - 0,5 < m_1 + 0,5 < m_2 - 0,5 < m_2 + 0,5 < \cdots < m_k - 0,5 < m_k + 0,5)$;
(iii) on dessine l'histogramme (le « diagramme en bâtons des fréquences ») à l'aide de la commande :

```
plt.hist(x,c,density='True',edgecolor='k',color='...', label="...")
```

où l'on a ajouté les options de tracé suivantes **(non exigibles)** :

- normalisation des rectangles (la surface totale vaut 1) : density='True'
- contours des rectangles en noir : edgecolor='k'
- couleur des rectangles : color='...' (mettre le nom de la couleur en anglais)
- légende associée à chaque histogramme : label="..." (mettre la légende choisie)

Remarque. Pour réaliser plusieurs graphiques dans une même fenêtre et ainsi pouvoir mieux les comparer, on peut utiliser l'instruction `plt.subplot(n,m,k)` avant chaque instruction de tracé de graphique, qui découpe la fenêtre graphique en n ligne et m colonnes, k indiquant le numéro de la colonne souhaitée pour chaque graphique.(Voir exemple cours en 2ème année)

10 Représentation graphique d'une fonction de deux variables.

Le graphe d'une fonction de deux variables $(x,y) \longmapsto f(x,y)$ définie sur un ouvert U est la surface S_f de l'espace formée de tous les points $M \begin{pmatrix} x \\ y \\ f(x,y) \end{pmatrix}$ lorsque (x,y) décrit U.

Afin de représenter une fonction de deux variables à l'aide de Python, nous aurons besoin d'importer les librairies suivantes :

```
import numpy as np   # que vous connaissez très bien
import matplotlib.pyplot as plt
```

Nous aurons également besoin de la fonction `Axes3D` de la librairie `mpl_toolkits.mplot3d,` qu'on importe de la façon suivante :

```
from mpl_toolkits.mplot3d import Axes3D
ax=Axes3D(plt.figure())
```

10.1 Définition.

Soient x, y des vecteurs de taille respective n et m. L'instruction

```
X,Y = np.meshgrid(x,y)
```

permet de construire le maillage $((x_i, y_j))_{(i,j)\in[\![1,n]\!]\times[\![1,m]\!]}$. Pour tracer la représentation graphique de f sur$[a,b]\times[c,d]$, on procèdera comme suit :

- On crée deux vecteurs x et y découpant les intervalles $[a,b]$ et $[c,d]$ en n petits intervalles de même longueur comme suit :

```
x=np.linspace(a,b,n)
y=np.linspace(c,d,n)
```

- On crée ensuite un maillage $((x_i, y_j))_{1\leq i,j\leq n}$du domaine $[a,b]\times[c,d]$ avec la commande :

```
X,Y = np.meshgrid(x,y)
```

- On trace avec l'instruction :

```
ax.plot_surface(X,Y,f(X,Y))
plt.show()
```

10.2 Exemple

Soit la fonction f définie sur : $[-1,1]$ par $f : (x,y) \longmapsto x \times y$.

```
n=21
def f(x,y):
    return x*y
```

On créer un maillage $((x_i, y_j))_{1\leq i,j\leq n}$ du domaine $D = [-1,1] \times [-1,1]$ par les instructions de la **définition 1.2** puis représenter la fonction sur le domaine D de la façon suivante

```
x=linspace(-1,1,n)
y=x
X,Y =np.meshgrid(x,y)
ax.plot_surface(X,Y,f(X,Y),cmap='jet')
plt.show()
```

- Les commandes `plt.contour(X,Y,f(X,Y),N)` ou `plt.contour(X,Y,f(X,Y),T)` tracent les lignes de niveau de la fonction f
- La commande `plt.quiver(X,Y,dX,dY)` trace en chaque point $(X[i], Y[j])$ du plan le vecteur de coordonnées $(dX[i], dY[j])$ pour tout $(i,j) \in [\![1,n]\!] \times [\![1,m]\!]$.
- On utilise `plt.quiver(X,Y,dX,dY) pour` tracer le vecteur gradient $\nabla f(x_i, y_j)$ en Python.
- Pour plus de détaille suu ses deux derniers points, voir le cours sur les fonctions à plusieurs variables(Feuille 5) .

Vos observations :