Programmes à connaître

1. Demander des entrées et afficher des résultats avec print, input et str

Exemple : écrire un programme qui calcule le discriminant d'une équation du second degré du type $ax^2 + bx + c$.

```
print("Ce programme calcul le discriminant de  ax2+bx+c=0")
a=float(input('Donner la valeur de a:'))
b=float(input('Donner la valeur de b:'))
c=float(input('Donner la valeur de c:'))
d=b**2-4*a*c
print('Le discriminant vaut '+str(d)+'.')
```

1. **print** affiche une chaîne de caractère, un résultat.
2. **input** permet l'affectation à une variable d'une valeur rentrée par l'utilisateur.
3. **str** permet de transformer le contenu d'une variable en chaine de caractère.
4. **float** est utilisé pour renvoyer un nombre à virgule flottante à partir d'un nombre ou d'une chaîne de caractère.

2. Définir une fonction

Exemple : définir la fonction $f : x \mapsto |\ln x + \sqrt{x} + e^x|$.

```
import numpy as np   #importation du module contenant les fonction usuelles
def f(x):
    y= np.abs(np.log(x)+ np.sqrt(x)+np.exp(x))
    return y
```

On peut aussi écrire directement

```
import numpy as np   #importation du module contenant les fonction usuelles
def f(x):
    return np.abs(np.log(x)+ np.sqrt(x)+np.exp(x))
```

Attention à l'indentation qui est fondamentale ici !

3. Représentation graphique

Tracé simple.

Exemple : représenter la fonction exponentielle sur l'intervalle $[-2; 2]$ avec un pas de 0.1.

```
import numpy as np
import matplotlib.pyplot as plt  #importation du module permettant les représentation graphiqu
x=np.arange(-2,2.1,0.1)         #création d'un vecteur allant de -2 à 2 avec un pas de 0.1
y= np.exp(x)                #calcul du vecteur image par l'exponentielle
plt.plot(x,y)               #tracé du graphe
plt.show()                  # affichage du graphe
```

Attention, `np.arange(a,b,c)` crée un vecteur dont les coefficients vont de a à b (**attention non inclus**, autrement dit on s'arrête à $b-c$) avec un pas de c.

Tracé multiple.

Exemple : représenter avec deux couleurs différentes sur l'intervalle $[-2;2]$ avec un pas de 0.1, la fonction exponentielle et sa tangente au point d'abscisse 0.
Ajouter un titre et une légende.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-2,2.1,0.1)
y1= np.exp(x)
y2= x+1
plt.plot(x,y1,'b', label="fonction exponentielle")
plt.plot(x,y2,'g', label="tangente en 0")
plt.title('Fonction exponentielle et sa tangente en 0')
plt.legend()    #affiche la légende définie par label=..."
plt.show()
```

4. Calcul de sommes avec une boucle for

Exemple : calculer $\sum_{k=0}^{n} k$

```
n=int(input('Donner une valeur de n: '))
S=0
for k in range(1,n+1):  # k in range(1er indice, dernier indice+1)
   S=S+k                #S=S+term général de la somme
print("S= ", S)
```

Remarques :

1. Attention à l'indentation pour les boucles for.
2. On n'oublie pas d'**initialiser** la somme à 0.
3. Si l'on souhaite sommer de 2 à n, on remplacera `range(1,n+1)` par `range(2,n+1)`.
4. Si l'on souhaite calculer $\sum_{k=0}^{n} k^2$, on remplacera `S=S+k` par `S=S+k**2`.
5. On retiendra que la commande `range(a,b,c)` crée une séquence d'entiers allant de `a` à `b` (**attention non inclus** c'est à dire que le dernier entier est `b-c`) avec un pas de `c`.
   Ainsi si l'on veut les entiers allant de `a` à `b` inclus avec un pas de `c`, on écrit `range(a,b+c,c)`.

A noter que `a, b, c` doivent être des entiers nécessairement.
Si `a` et `c` ne sont pas préciser alors par convention `a=0` et `c=1`.
Ainsi `range(4)` crée la séquence des entiers consécutifs entre 0 et 3 et celle `range(1,5)` crée ma séquence des entiers consécutifs entre 1 et 4.

5. Calculer le n-ème terme d'une suite définie par relation de récurrence simple

On utilisera une boucle for.

Exemple : Soit (u_n) une suite définie par $u_0 = 4$ et $\forall n \in \mathbb{N},\ u_{n+1} = 6u_n + 1$.
Ecrire une procédure qui calcule u_n pour une valeur de n donnée.

```
n=int(input('Entrer un entier n:'))
u=4
for i in range(1,n+1):
   u=6*u+1
print("u(",n,")=",u)
```

Remarques :

- Si le premier terme vaut autre chose que 4, on écrira bien sûr la valeur de u_0 à la place de 4.
- Et si le premier terme de la suite est u_1 et non u_0, on écrira dans la boucle `for i in range(2,n+1):` à la place de `for i in range(1,n+1):` (puisque le premier terme étant u_1 le terme suivant à calculer dans la boucle est u_2, c'est à dire celui d'indice 2).
- De manière générale si $u_{n+1} = f(u_n)$, on remplacera dans la boucle `6*u+1` par `f(u)`, où `f` est une fonction à définir au préalable.

Ecrire une procédure qui représente à la volée les premiers termes de la suite jusqu'à celui de rang n.

```
n=int(input('Entrer un entier n:'))
u=4
for i in range(1,n+1):
   u=6*u+1
   plt.plot(i,u, 'x')
plt.show()
```

6. Structures conditionnelles

Les opérateurs de comparaisons :

$=$	`==`
$\neq$	`!=`
$<$	`<`
$>$	`>`
$\leq$	`< =`
$\geq$	`> =`

Attention, l'égalité se note == et non = qui sert à l'affectation !
Les trois types de structures conditionnelles.

- if..
- if...else...
- if...elif...else

Exemple :

Définir la fonction $f : \begin{cases} x-1 & \text{si } x \geq 1 \\ 0 & \text{si } -1 < x < 1 \\ x+1 & \text{si } x \leq -1 \end{cases}$.

```
def f(x):
    if x>=1:
        y=x-1
    elif -1<x<1:
        y=0
    else:
        y=x+1
    return y
```

7. Calculer le n-ème terme d'une suite définie par relation de récurrence double

On utilisera aussi une boucle for.
Mais attention ici il faudra faire intervenir une variable auxiliaire pour pouvoir conserver à chaque fois les deux termes précédents de la suite.

Exemple : soit (u_n) une suite définie par $u_0 = 0,\ u_1 = 1$ et $\forall n \in \mathbb{N},\ u_{n+2} = u_{n+1} + u_n$.
Ecrire une procédure qui calcule u_n pour une valeur de n donnée.

```
n=int(input('donner la valeur de n:'))
u=0
v=1       //initialisation des deux premiers termes
for k in range(2,n+1):
   w=u+v // variable "mémoire"
   u=v
   v=w
print("le terme de rang ", n, " vaut ", v)
```

8. Calcul du premier rang d'une suite dépassant un seuil

Soit (u_n) une suite définie par u_0 et $\forall n \in \mathbb{N},\ u_{n+1} = f(u_n)$ tel que $\lim\limits_{n\to+\infty} u_n = +\infty$.
Nous allons voir un programme qui permet de déterminer le premier rang n tel que u_n dépasse un seuil donnée.

Exemple : Soit (u_n) une suite définie par $u_0 = 1$ et $\forall n \in \mathbb{N},\ u_{n+1} = 1 + u_n^2$.
Ecrire une procédure qui calcule le premier rang n à partir duquel $u_n > 1000$ et représente tous les termes calculés (rmq :cela à du sens car (u_n) diverge vers $+\infty$).

```
u=1
n=0
seuil=1000
while u<= seuil:
    n=n+1
    u=1+u**2
    plt.plot(n,u,'x')
```

```
print("le rang vaut ", n)
plt.show()
```

- **Attention à la condition dans la boule while.**
  Si on souhaite trouver le premier rang n à partir duquel u_n>1000, on continue à calculer les termes tant que u_n≤1000.
- De façon plus générale, si $u_{n+1} = f(u_n)$, on peut définir la fonction f au préalable et remplacer `u=1+exp(u)` par `u=f(u)`.

9. Calcul du premier rang d'une suite convergente approchant sa limite avec une precisi

Soit (u_n) une suite définie par u_0 et $\forall n \in \mathbb{N},\ u_{n+1} = f(u_n)$ telle que $\lim\limits_{n\to+\infty} u_n = \ell$.
Nous allons voir un programme qui permet de déterminer le premier rang n tel que u_n approche la limite ℓ avec une précision donnée.

Exemple :
Soit (u_n) une suite définie par $u_0 = 14$ et $\forall n \in \mathbb{N},\ u_{n+1} = 6 - 0.5u_n$.
Ecrire une procédure qui calcule le premier rang n à partir duquel $|u_n - 4| < 10^{-3}$ et représenter les termes calculés (rmq : cela à du sens car (u_n) converge vers 4).

```
import numpy as np
pre=10**(-3)
u=14
n=0
while np.abs(u-4)>=pre:
    n=n+1
    u=6-0.5*u
    plt.show(n,u,'x')
print("Le rang vaut ", n)
plt.show()
```

- **Attention à la condition dans la boule while.**
  Si on souhaite trouver le premier rang n à partir duquel $|u_n-4|$<10^{-3}, on continue à calculer les termes tant que $|u_n - 4|$>=10^{-3}.
- De façon plus générale, si $u_{n+1} = f(u_n)$, on peut définir la fonction f au préalable et remplacer `u=6-0.5*u` par `u=f(u)`.

10. Listes

Commandes à connaître

L[i]	élément n°i de la liste L **(attention la numérotation commence à 0)**
L.append(i)	ajout de l'élément i à la fin de la liste L
len(L)	nombre d'éléments de la liste L
x in L	renvoie True si x est un élément de L, False sinon
del(L[i])	supprime l'élément n°i de la liste L
L.count(x)	compte le nombre d'occurrence de x dans L
L+M	concaténation de la liste L avec la liste M
L*m	ajoute m fois L à elle-même
list(range(a,b,c))	crée une liste avec les **entiers** allant de a à b-c avec un pas de c
max/min(L)	renvoie le maximum/minimum des éléments de L

Attention, à bien noter que `L+M` **n'est pas la somme** de `L` et de `M`, mais la **concaténation** des deux listes.
L'addition de deux listes n'a pas de sens car les coefficients des listes ne sont pas nécessairement des valeurs numériques.

Exemple :
Créer la liste qui contient les éléments 1, 2, chien, 4 , 2 ,3, lapin, 5, 2 et 7.
Écrire une commande qui renvoie le nombre d'élément de la liste.
Écrire une commande pour récupérer l'élément 1, puis l'élément 4.
Ajouter 'chat' à la liste L.
Écrire une commande qui compte le nombre d'occurrence de l'élément 2 dans la liste L.
Supprimer l'élément 5 de la liste L.
Écrire une commande qui renvoie True si 'chien' fait partie de la liste et 0 sinon.

```
L=[1,2,'chien', 4 ,2 ,3, 'lapin',5,2,7]
len(L)
L[0]
L[3]
L.append('chat')
L.count(2)
del(L[7])
'chien' in L
```

Exemple :
Que renvoie les commandes `range(5)`, `range(1,5)` et `range(1,10,2)`.

Réponse :
`range(5)` *renvoie la séquence des entiers 0,1,2,3,4*
(par convention si un seul paramètre est précisé il s'agit de b et considère que l'on part de 0 et que le pas est égal 1).
`range(1,5)` *renvoie la séquence 1,2,3,4*
(par convention si le pas n'est pas précisé il considéré comme égal à 1).
`range(1,10,2)` *renvoie la séquence 1,3,5,7,9*

Exemple :
Définir la liste des entiers allant de 1 à 10 au carré à l'aide d'une boucle for.
Refaire la même chose avec une liste en compréhension.

```
L = []
for k in range(1,11):
     L.append(k**2)

L = [k**2 for k in range(1,11)]
```

Exemple :
Écrire un programme calculant le nombre d'occurence d'un élément x dans une liste L.

Solution :

```
def occurence(L,x):
     occ = 0
     for e in L:
         if e==x:
             occ=occ+1
     return occ
```

Exemple :
Écrire un programme permettant de créer une liste L contenant les termes de rang 0 à n de la suite (u_n) définie par $u_0 = 5$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = 2u_n + 3$.
Solution :

```
n=int(input("Nombre de termes à calculer?')
u=5
L=[u]
for i in range(1,n+1):
    u=2*u+3
    L.append(u)
```

Exemple :
Créer une liste L de 20 nombres entiers tirés au hasard entre 1 et 50, puis créer une liste I contenant les indices de la liste L pour lesquels l'élément associé est > 35 et l'afficher.
Solution :

```
import random as rd
L=[]
for i in range(20):
    nb=rd.randint(1,50)
    L.append(nb)

I=[]
for j in range(len(L)):
    if L[j]>35
    I.append(j)
print(I)
```

11. Statistiques descriptives

Soit T un tableau une ligne contenant un échantillon d'un caractère quantitatif relevé sur une population.

On retiendra les commandes suivantes permettant décrire cet échantillon et qui nécessitent soit le package numpy (quand elles commencent par np), soit le package matplotlib.pyplot (quand elles commencent par plt) :

T=np.array([, , ,])	création d'un tableau une ligne T
len(T)	longueur du tableau T
np.median(T)	médiane de T
np.quantile(T, 0.25)	1er quartile de T
np.quantile(T, 0.75)	3ème quartile de T
np.min(T)	minimum de T
np.max(T)	maximum de T
np.mean(T)	moyenne de T
np.var(T)	variance de T
np.std(T)	écart-type de T
np.cumsum(T)	somme cumulées de T
plt.boxplot(T)	boîte à moustache de T
plt.bar(valeurs, effectifs , width=0.5)	diagramme en bâton des effectifs de la série stat. issue de T où valeurs représente un tableau une ligne contenant les modalités de la série

Exemple :
Le tableau suivant donne les notes des élèves d'une classe.

Elèves	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
Notes	15	10	12	8	10	18	12	8	8	15	10	8	6	18	12	8	12

La série statistique associée est

Notes x_i	6	8	10	12	15	18
Effectifs n_i	1	5	3	4	2	2

1. Écrire les commandes Python permettant de définir la série de notes ci-dessus, d'en calculer l'effectif, la médiane, le 1er et 3ème quartile, l'étendue, la moyenne et la variance.
2. Ajouter les commandes permettant de définir le tableau des effectifs, puis de calculer les effectifs cumulés et les fréquences cumulées à partir de ce dernier.
3. Écrire un programme Python permettant de tracer le diagramme en bâton, ainsi que la boîte à moustache associée à cette série.

Solution :

1.
```
#importation des packages
import numpy as np
import matplotlib.pyplot as plt

#calcul
T=np.array([15, 10,12,8,10,18,12,8,8,15,10,8,6,18,12,8,12])
effectif_total=len(T)
mediane=np.median(T)
Q1=np.quantile(T, 0.25)
Q3=np.quantile(T, 0.75)
etendue=np.max(T)-np.min(T)
moyenne=np.mean(T)
variance=np.var(T)
```

```
#affichage
print('L\'effectif total  vaut '+str(effectif_total))
print('La médiane vaut '+str(mediane))
print('Le premier quartile vaut '+str(Q1))
print('Le troisième quartile vaut '+str(Q3))
print('L\'étendue vaut '+str(etendue))
print('La moyenne vaut '+str(moyenne))
print('La variance vaut '+str(variance))
```

2.
```
effectifs=np.array([1,5,3,4,2,2])
effectifs_cumules=np.cumsum(effectifs)
frequences_cumules=effectifs_cumules/effectif_total
print('Le tableau des effectifs cumulés: ' +str(effectifs_cumules))
print('Le tableau des fréquences cumulées: ' +str(frequences_cumules))
```

3.
```
#diagramme en baton
valeurs=np.array([6,8,10,12,15,18])
effectifs=np.array([1,5,3,4,2,2])
plt.bar(valeurs, effectifs, width=0.5)
plt.title('Diagramme en bâton de la série de notes')
plt.show()

#boîte à moustache
plt.boxplot(T)
plt.title('Boîte à moustache de la série de notes')
plt.show()
```

12. Recherche d'une occurrence par dichotomie

Complétez le programme suivant permettant de rechercher une occurrence donnée x dans une liste L triée et de renvoyer 'Trouvé' si l'occurrence existe ainsi que le rang où se trouve cette occurrence et 'Non trouvé' sinon :

```
def occurence_triée(L,x):
    a = 0
    b= len(L)-1
    while a <= b:
        m = ...
        valeur_m=L[m]
        if valeur_m==x:
            return 'Trouvé à l'indice'+srt(m)
        elif valeur_m<x:
            a=...
        else:
            b=....
    return 'Non trouvé'
```

Solution :

```
def occurence_triée(L,x):
    #indice de début de la liste initiale
    a = 0
    #indice de fin de la liste initiale
    b= len(L)-1
    #boucle de recherche tant que liste non vide
    while a <= b:
        #calcul du 'milieu' (quotient par la div
         eucli. par 2)
        m = (a+b)//2
        #valeur du 'milieu'
        valeur_m=L[m]
        #comparaison avec la valeur à chercher
        if valeur_m==x:
            return 'Trouvé à l'indice'+srt(m)
        elif valeur_m<x:
            #on garde la partie droite de la liste
            a= m+1
            #on garde la partie gauche de la liste
        else:
            b= m-1
    #retourne "Non trouvé" si x non trouvé
    return 'Non trouvé'
```

13. Vecteurs et matrices

Création d'un vecteur :

Créer le vecteur $x = (1, 2, 3, 5)$

```
x= np.array([1,2,3,5])
```

Création d'une matrice :

Créer la matrice $A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$

```
A= np.array([[1, 2, 3, 4], [5,6,7,8]])
```

Vecteurs particuliers :

Commande	Signification
np.arange(a,b,c)	vecteur dont les coefficients vont de a à b (non inclus) avec un pas de c contrairement à range les coefficients de np.arange ne sont pas nécessairement des entiers
np.linspace(a,b,c)	vecteur contenant c coefficients équirépartis entre a et b
np.zeros(n)	vecteur nul de longueur n
np.ones(n)	vecteur de longueur n contenant que des 1

Matrices particulières :

Commande	Signification
np.zeros([n,p])	matrices de n lignes et p colonnes avec tous les coefficients égaux à zéro
np.ones([n,p])	matrices de n lignes et p colonnes avec tous les coefficients égaux à un
np.eye(n)	matrice identité de taille n

Extraction de coefficients

Attention les indices des lignes et colonnes commencent à 0.

Soit T une matrice.

Opérateur	Signification
T[i,j]	élément à la i-ème ligne et j-ème colonne
T[i, :]	ligne i
T[:,j]	colonne j
T[i :j, k :l]	sous-matrice de T, de la i-ème à la **j-1-ème** ligne et de la k-ème à la **l-1-ème** colonne
T[i :j, :]	sous-matrice de T, de la i-ème à la **j-1-ème** ligne, toutes les colonnes incluses
T[: ,i :j]	sous-matrice de T, de la i-ème à la **j-1-ème** colonnes, toutes les lignes incluses

Opérations purement matricielles :

Soit T et M deux matrices et a un réel.

A condition que les opérations aient du sens, voici les opérations classiques sur les matrices :

Opérateur	Signification
T + M	somme coefficients par coefficients des tableaux T et M (qui doivent donc être de même taille)
a*T	multiplication de tous les coefficients de T par a
np.dot(T, M)	produit matriciel de T par M
np.transpose(T)	transposée de T
np.inv(T)	inverse de T

Opérations terme à terme :

Soit T et M deux matrices de même taille et a un réel.

Opérateur	Signification
T*M	produit des coefficients de T par ceux de M
T/M	division des coefficients de T par ceux de M
T+a	ajout de a à tous les coefficients de T
T*a	multiplication de tous les coefficients de T par a
T/a	division de tous les coefficients de T par a
T**a	tous les coefficients de T sont élevés à la puissance a

Attention à ne pas confondre `T*M` (produit terme à terme) et `np.dot(T,M)` (produit matriciel).

Autres commandes :

Soit M une matrice, B un vecteur et n un entier.

Autres opérateurs	Signification
np.shape(M)	taille de la matrice
np.shape(M)[0]	nombre de lignes de la matrice
np.shape(M)[1]	nombre de colonnes de la matrice
np.f(M)	applique la fonction `f` aux coefficients de M `f` peut être remplacé par `exp, log, sqrt, prod, sum, cumsum, mean, max, min...`
np.f(M,0)/ np.f(M,1)	applique la fonction `f` aux lignes/ colonnes de M `f` peut être remplacé par `prod, sum, cumsum, mean, max,min...`
al.matrix_power(M,n)	puissance nième de M (nécessite d'importer le package numpy.linalg as al)
np.linalg.solve(M, B)	résout le système MX=B (nécessite d'importer le package numpy.linalg as al)

Attention à ne pas confondre `M**n` (coefficients de M élevés à la puissance n) et `al.matrix_power(M,n)` (calcul de M^n).

Exercices

Exemple :

Écrire un programme qui crée la matrice $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

Solution :

```
A= np.array([[1, 2, 3], [4, 5, 6],  [7,8,9]])
```

Exemple :

Écrire un programme permettant de calculer $\sum_{k=1}^{100} k^2$ **sans passer par une boucle for**.

Solution :

```
import numpy as np
x=np.arange(1,101,1)
S=sum(x**2)
```

Exemple :
Écrire un programme permettant de créer la matrice sans renter les coefficients à la main $A = \begin{pmatrix} 2 & 3 & 3 \\ 3 & 2 & 3 \\ 3 & 3 & 2 \end{pmatrix}$.

et $B = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$.

Solution :

```
import numpy as np
A= 3*np.ones([3,3])-np.eye(3)

B=np.ones([4,4])
B[1:3,1:3]=np.zeros([2,2])
```

Exemple :
Écrire un programme Python permettant de créer la matrice $M = ((m_{ij}))_{1 \leq i \leq 3, 1 \leq j \leq 5}$, où $m_{ij} = i + j$.

Solution :

```
M=np.array([[i+j for j in range(1,6)] for i in range(1,4)])
```

14. Théorie des graphes

Exercice :

1. Écrire un programme qui, étant donné la matrice d'adjacence A d'un graphe, renvoie l'ordre du graphe.
2. Écrire une fonction Python qui, étant donnée une matrice d'adjacence A d'un graphe à n sommets (numérotés de 1 à n) et un sommet i, renvoie le degré du sommet i.
3. Écrire un programme qui, étant donné la matrice d'adjacence A d'un graphe connexe, renvoie True si le graphe est eulérien et False sinon.
4. Écrire un programme qui, étant donné la matrice d'adjacence A d'un graphe à n sommets (numérotés de 1 à n), ainsi que deux sommets i et j du graphe et renvoie le nombre de chemin de longueur k (compris entre 1 et n) reliant i à j.
5. Écrire un programme qui, étant donné la matrice d'adjacence A d'un graphe à n sommets (numérotés de 1 à n) renvoie True si le graphe est connexe et False sinon.

Solution :
On commencera par importer les packages `numpy` et `numpy.linalg` en tapant les commandes suivantes :
`import numpy as np import numpy.linalg as al`

1. ```
   def ordre(A):
       y=np.shape(A)[0]
       return print("l'ordre du graphe est",  y)
   ```

   En effet, l'ordre du graphe correspond au nombre de sommets du graphe et le nombre de sommets du graphe correspond au nombre de ligne (ou colonne) de la matrice d'adjacence (commande `np.shape`*.*
   ```

2.
```
def degre(A, i):
    y=sum(A[i,:])
    return print('le degré du sommet', i, 'vaut',  y)
```

En effet, le degré du sommet n° i (en numérotant les sommets à partir de 0) correspond au nombre d'arêtes qui en part, ce qui correspond à la somme de la ième ligne de la matrice d'adjacence.

3.
```
def graph_euler(A):
    n=np.shape(A)[0]
    for s in range(0,n):
        if degre(A,s) \% 2==1:
            return False
    return True
```

En effet, nous avons vu dans le cours que pour un graphe connexe, il y avait équivalence entre le fait que le graphe soit eulérien et le fait que tous les sommets soient de degré pair. De ce fait, dans ce programme, on passe en revue chacun des sommets, on calcule son degré grâce au programme précédent et si on trouve un sommet de degré impair alors on renvoie False (le graphe ne peut être eulérien). Autrement, on renvoie True (le graphe est bien eulérien, car si on ne trouve aucun sommet de degré impair, c'est que tous sont de degrés pairs). Pour tester si le degré d'un sommet s est impair, on calcule son reste de la division euclidienne par 2 (commande %2) et si on trouve qu'il vaut 1 alors cela signifie que le degré du sommet s est impair. Sinon ce reste 0 et dans ce cas le degré du sommet s est pair.

4.
```
def nb_chemin(A,i,j,k):
    M=al.matrix_power(A,k)
    y=M[i,j]
    return print('le nombre de chemin de longueur', k, 'entre les sommets', i, 'et',
     j, 'vaut', y)
```

En effet, nous avons vu dans le cours que le nombre de chemin de longueur k entre deux sommets i et j correspondait au coefficient situé à la $i$ème ligne et $j$ème colonne de la matrice A^k (où A est la matrice d'adjacence du graphe) . D'où, le fait que l'on calcule la matrice $M = A^k$ grâce à la commande `M=al.matrix_power(A,k)` *et que l'on récupère ensuite son coefficient situé en position (i, j) (commande* `M[i,j]` *).*

5.
```
def graph_connexe(A):
    n=np.shape(A)[0]
    M=np.eye(n)
    for i in range(1,n):
        M=M+al.matrix_power(A, i)
    nb_termes_nuls= np.sum(M==0)
    if nb_termes_nuls >=1:
        return False
    else:
        return True
```

En effet, nous avons vu dans le cours qu'un graphe est connexe si et seulement si tous les coefficients de la matrice $I + A + ... + A^{n-1}$ sont strictements positifs (où A est la matrice d'adjacence du graphe et n son son ordre).
D'où, on part de la matrice I (commande `M=np.eye(n)`*), puis à l'aide d'un boucle for on calcule la matrice $I + A + ... + A^{n-1}$ que l'on note M. Ensuite, on compte le nombre de*

coefficients nuls de la matrice M (commande `np.sum(M==0)`*) et si ce nombre est supérieur ou égal à 1 (ce qui signifie qu'il y a au moins un coeff nul et donc que tous les coeffs ne sont pas strictement positifs) alors on renvoie False. Sinon (dans ce cas tous les coeffs sont strictement positifs) on renvoie True.*

15. Algorithmes gloutons

Définition

Un algorithme glouton (*greedy algorithm*) est un algorithme qui consiste à optimiser localement le problème, étape par étape, c'est à dire à choisir des solutions locales optimales d'un problème dans le but d'obtenir une solution optimale globale au problème.
Au cours de la construction de la solution, l'algorithme résout une partie du problème puis se focalise ensuite sur le sous-problème restant à résoudre.
L'avantage de ces algorithmes c'est qu'ils sont faciles à mettre en oeuvre et souvent beaucoup moins coûteux en temps que la méthode brute. Leur principal défaut c'est qu'ils ne renvoient pas toujours la meilleure solution.

Exemples

1. le problème du rendu de monnaie (donner une somme avec le moins possible de pièces) : l'algorithme consistant à répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante est un algorithme glouton,
2. le problème de l'allocation de salles de cours,
3. le problème du plus court chemin dans un graphe (algorithme de Dijkstra).

Algorithmes de Djikstra (prononcé "Dextra")

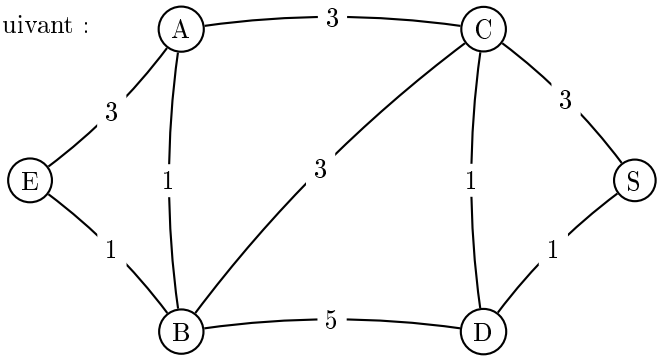
Cet algorithme permet, dans un graphe pondéré positivement, de trouver la chaîne de poids minimal menant d'un sommet donné à un autre (exemple d'utilisation : plus court chemin pour se rendre d'une ville à une autre connaissant le réseau routier d'une région).

Voici un lien vers un site où tous cela est encore plus détaillé sur un exemple :
https://www.maths-cours.fr/methode/algorithme-de-dijkstra-etape-par-etape.
Et une video.

Étape	Tâches à effectuer
1	Sur la deuxième ligne du tableau, écrire le coefficient 0 sous le sommet de départ et ∞ sous les autres sommets.
2	Tant qu'il y a des sommets non sélectionnés : 1. Sur la dernière ligne écrite, repérer et sélectionner le sommet X de coefficient minimal 2. Commencer une nouvelle ligne et rayer toutes les cases vides sous X. 3. Pour chaque sommet Y adjacent à X : (a) calculer la somme P du coefficient de X et du poids de l'arête reliant X à Y ; (b) si P est strictement inférieur au coefficient actuel de Y, inscrire P dans la case correspondante de la colonne Y en indiquant à droite entre parenthèse sa provenance ; (c) sinon, reporter l'ancien coefficient de Y et sa provenance. 4. Pour les sommets non adjacents reporter les coefficients de la ligne précédente.
3	1. La longueur minimale est le nombre lu sur la dernière ligne du tableau. 2. Les sommets inscrits entre parenthèses permettent par remontée de trouver la chaîne minimale.

Exemple :

Considérons le graphe suivant :



Quelle est la plus courte chaîne entre les sommets E et S ?

Solution :

Voici ce que donne l'algorithme de Djikstra sur le graphe précédent :

E	A	B	C	D	S	choix	coef.
0	∞	∞	∞	∞	∞	E	0
	$0+3=3$ (E)	$0+1=1$ (E)	∞	∞	∞	B	1
	$1+1=2$ (B)		$1+3=4$ (B)	$1+5=6$ (B)	∞	A	2
			2+3=5>4 on garde 4 (B)	6 (B)	∞	C	4
				$4+1=5$ (C)	$4+3=7$ (C)	D	5
					$5+1=6$ (D)	S	6

On obtient comme chaîne E-B-C-D-S de poids 6.

16. Le programme de la dichotomie

Le programme de la dichotomie permet d'obtenir la valeur approchée à une précision donnée de la solution α de l'équation $f(x) = 0$, où f est une fonction continue qui s'annule en une unique valeur sur un intervalle $[x; y]$.

On retiendra les deux programmes suivants :

1. Soit **on fixe le nombre n d'itérations de l'algorithme**.

```
def dicho(x,y,n):
    a=x
    b=y
    for k in range(1,n+1):
        m=(a+b)/2
        if f(a)*f(m) < 0:
            b=m
        else:
            a=m
    return [a,b]
```

2. Soit **on fixe une précision de l'encadrement qu'on veut obtenir**.

```
def dicho(x,y,eps):
    a=x
    b=y
    while(b-a>eps):
        m=(a+b)/2
        if f(a)*f(m) < 0:
            b=m
        else:
            a=m
    return [a,b]
```

17 Commande rand et simulation de phénomènes aléatoires

On retiendra que :

- `rd.random()` retourne un **nombre tiré aléatoirement et de manière uniforme dans l'intervalle** $[0, 1[$.
- `rd.random(n)` retourne un vecteur de longueur n dont les coefficients sont des **nombres tirés aléatoirement et de manière uniforme et indépendante** dans l'intervalle $[0, 1[$.
- l'évènement $U < p$, où $U =$`rd.random()`, est un évènement de probabilité p.

Exemple : simulation d'un lancer de dé

Écrire une commande Python permettant de simuler un lancer de dé, en utilisant la commande `rd.random()`, puis `rd.randint`.

Solution :

```
import numpy as np
import numpy.random as rd
```

```
np.floor(rd.random()*6)}+1  #à savoir expliquer
rd.randint(1,7)  #attention rd.randint(a,b)= réalisation d'une uniforme sur [|a, b-1|]
```

Exemple : simulation d'un lancer de pièce

On lance une pièce telle que la probabilité de faire pile soit égale à 1/3. Proposer une suite d'instructions qui permet de simuler ce lancer de pièce.

Solution

```
p=1/3
U=rd.random()
if U<p:
    print('Pile')
else:
    print('Face')
```

La plupart des lois de probabilités usuelles et la commande générale prend la forme suivante : `rd.nomdelaloi(*)`, où `*` représente les paramètres de la loi.

Commande	**Signification**
rd.random()	nombre tiré au hasard entre 0 et 1
rd.randint(a,b)	entier tiré au hasard entre a et b-1
rd.random(N)	N réalisation d'une loi $\mathcal{U}([0,1])$
rd.binomial(n,p,N)	N réalisations d'une loi $\mathcal{B}(n,p)$
rd.randint(a,b,N)	N réalisations d'une loi $\mathcal{U}([\|a,b-1\|])$
rd.geometric(p,N)	N réalisations d'une loi $\mathcal{G}(p)$
rd.poisson(λ, N)	N réalisations d'une loi $\mathcal{P}(\lambda)$

TABLE 1 – Lois de probabilités usuelles

Si le nombre de réalisations N à effectuer n'est pas renseigné alors par défaut une seule réalisation sera faite. Au contraire si l'on souhaite créer un tableau ayant L lignes et C colonnes dont les coefficients seraient des réalisations d'une loi donnée alors il faut remplacer N par [L, C].

18. Histogramme-loi empirique et théorique

La commande `plt.hist(valeurs, bins= *)` permet de tracer un histogramme, elle prend deux arguments :

- le **vecteurs** `valeurs` qui contient la série de données dont on veut tracer l'histogramme ;
- le **paramètre** `bins` permettant de définir les classes de l'histogramme :
  — soit en donnant le nombre n de classes en rentrant un entier. Dans ce cas Python se charge alors de découper l'intervalle allant de la plus petite valeur à la plus grande. Par exemple, avec `bins=6`, on aura 6 classes ;

— soit en donnant le nombre de classes et leurs largueurs, en rentrant une liste, un tableau ou une séquence définissant les bornes des classes (valeur de gauche incluse et celle de droite exclue (sauf la dernière)).
Par exemple, avec `bins = [0,1,4,6]`, on aura trois classes d'intervalles de longueur inégales [0,1[; [1,4[et [4,6[.
Alors qu'avec `np.arange(-0.5, 5.5)` on aura 5 classes de longueurs égales [-0.5, 0.5[, [0.5, 1.5[, [1.5, 2.5], [2.5, 3.5[, [3.5, 4.5 [.

On peut y ajouter d'autres arguements comme :
— `label=''` si on veut ajouter une légende ;
— `normed= True` si on veut non pas que ce soit les effectifs qui soient représentés mais les fréquences ;
— `rwidth=..` si on veut modifier la largueur des barres de l'histogramme.

Exemple :
Représenter sur un même graphique l'histogramme de 1000 réalisations d'une loi $\mathcal{U}([|1,8|])$ et la loi théorique associée.

Solution :

```
#importation des packages
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as rd

#simulation de 1000 réalisation d'une loi U([|1,8|])
valeurs=rd.randint(1,9,1000)

#création de l'histogramme
#avec des classes de longueur 1 centrées sur #les entiers allant de 1 à 8

classes=np.arange(0.5,9.5)
plt.hist( valeurs , bins=classes, rwidth=0.8, normed=True, label='loi empirique')

#ajout de la loi théorique
x=np.arange(1,9)
loi=np.ones(8)/8
plt.bar(x,loi, width=0.1, color='r', label='loi théorique')

#représentation graphique
plt.title ( ' Comparaison loi théorique-loi empirique ')
plt.legend()
plt.show ()
```

19. Librairie Pandas

La bibliothèque `pandas` permet la manipulation de fichier.csv et de tables de données.

```
import pandas as pd
```

La lecture d'un fichier se fait grâce à la commande `pd.read_csv("nom du fichier", delimiter=",")`

où `delimiter` permet de spécifier le caractère utilisé pour délimiter les champs du fichier (généralement des ; ou des , mais cela peut être aussi des espaces, etc).

Exemple :

```
villes = pandas.read_csv("villes.csv", delimiter=";")
```

Voici les autres commandes à retenir :

Description générale :

Commande	**Signification**
nom_tableau.head()	affiche les cinq première lignes de la table
nom_tableau.shape	affiche la taille de la table
nom_tableau.columns	retourne la liste des champs (ce que représentent les colonnes)
nom_tableau.dtypes	affiche la liste des champs, avec le type de données correspondant

Commande	**Signification**
nom_tableau.describe()	donne un résumé statistique de l'ensemble de la table de données colonne par colonne (moyenne, écart-type, minimum, 1er, 2ème et 3ème quartiles, maximum), attention uniquement pour les champs numériques.
nom_tableau.mean()	renvoie la liste des moyennes pour chaque colonne numérique
nom_tableau.std()	renvoie la liste des écart-types pour chaque colonne numérique
nom_tableau.median()	renvoie la liste des médianes pour chaque colonne numérique
nom_tableau.count()	renvoie le nombre de valeurs pour chaque colonne numérique

Classement, sélection :

Commande	**Signification**
nom_tableau.sort_values('Nom_colonne')	classe la table suivant les valeurs croissantes de la ordre alphabétique dans le cas de lettres)
nom_tableau.sort_values('Nom_colonne', ascending=False)	classe la table suivant les valeurs décroissantes d (par ordre alphabétique dans le cas de lettres)
nom_tableau[condition nom_tableau['Colonne']]	affiche les lignes de la colonne intitulée 'Colonne'