Les algorithmes randomisés - LAPORTE Guillaume - NOYON

Essais préalables

Entrée [4]:

```
import numpy.random as rd
L=[1,2,3,4,5,6] # liste des element a tirer aleatoirement
proba = [0.1,0.15,0,0.15,0.3,0.3] # liste des probas correspondantes
rd.choice(L,p=proba) # renvoie un element de L aleatoirement suivant la loi de proba
```

Out[4]:

```
5
```

Entrée [5]:

```
import numpy as np
D=np.load("djibouti.npy")
```

Entrée [2]:

```
print(D)
```

```
[[   0.          290.99301545  794.00183127 ... 1852.35373049
  1624.75194088 1858.09261272]
 [ 290.99301545    0.          512.54097969 ... 1598.94551098
  1412.88752368 1649.18902517]
 [ 794.00183127  512.54097969    0.         ... 1331.29480435
  1288.37008374 1514.19696932]
 ...
 [1852.35373049 1598.94551098 1331.29480435 ...    0.
   440.55907953  444.22745405]
 [1624.75194088 1412.88752368 1288.37008374 ...  440.55907953
     0.          236.48918264]
 [1858.09261272 1649.18902517 1514.19696932 ...  444.22745405
   236.48918264    0.        ]]
```

Résolution

ALGORITHME PROBABILISTE – VOYAGEUR DE COMMERCE ET COLONIES DE FOURMIS

Entrée [4]:

```
# Paramètres de l'algorithmes : VARIABLES GLOBALES
```

Entrée [1]:

```
# Importation de la bibliothèque numpy
import numpy as np
import numpy.random as rd
```

L'algorithme fait ainsi intervenir un grand nombre de paramètres que l'on a regroupé dans le tableau suivant :

paramètre	valeur	
α	2	importance des phéromones dans le choix de la prochaine ville
β	1	importance de la distance dans le choix de la prochaine ville
γ	0.5	incitation à découvrir de nouvelles pistes
Q	1	quantité de phéromones déposées
ρ	0.1	évaporation des phéromones

Entrée [9]:

```
# Variables globales
D=np.load("djibouti.npy") # Matrice adjacence
n=len(D) # Nombre de villes
alpha =2
beta=5
gamma=0
Q=1
rho=0.1
Phi = np.ones((n,n))  # On remplit de 1 de la matrice des phéromones phi(i,j) = phéro
```

- écrire une fonction `prochaineVille(villeCourante, nonVues, Phi, D)` qui étant donnée la ville courante `villeCourante`, la liste `nonVues` des villes non visitées et les matrices `Phi` et `D` renvoie la prochaine ville où va aller la fourmi.

Entrée [3]:

```python
def prochaineVille(villeCourante, nonVues, Phi,D):
    """
    Entrées : --> villeCourante : Ville en cours
              --> nonVues : Villes non visitées
              --> Phi : Matrices donnant la quantité de phéromones ( 0<=Phi(i,j)<=n-1
              --> D : Matrices des poids entre les villes ( 0<=D(i,j))<=n-1 )

    Sortie : villeSuivante : Prochaine ville visitée par la fourmi
    """
    cNonVues=[] # Nouvelle Liste des villes non vue
    cPhi=[]     # Nouvelle liste des phéromones voisin de la ville courante
    cD=[]     # Nouvelle liste des distances entre la ville courante et ses voisines


    for i in nonVues:
        if i != villeCourante:
            cNonVues.append(i)                # Liste des prochaines villes, ville c
            cPhi.append(Phi[villeCourante][i]) # Matrices des phéromones des voisins
            cD.append(D[villeCourante][i])     # Matrices des distances entre la vill


    cPhi = np.array(cPhi) # On transforme cPhi en tableau exploitable par numpy
    cD = 1/np.array(cD)   # n(i,j)=1/d(i,j)
    total = gamma + (cPhi**alpha)*(cD**beta) ##
    total1 = np.sum(total)                   # px = gamma + phi**alpha*D**beta / Tota
    px = total / total1                      ##
    villeSuivante=rd.choice(cNonVues,p=px)


    return villeSuivante

#Test Fonction
#prochaineVille(3,[0,1,2,3,5,6,8,9,10,12],Phi,D)
```

- on peut alors facilement en déduire une fonction `tour(Phi,D)` qui procèdera de la manière suivante :
  - on choisit une ville de départ au hasard
  - on initialise une liste des villes non vue contenant toutes les villes sauf celle de départ
  - on initialise une liste `chemin` ne contenant que la ville de départ
  - **Tant qu'**on n'a pas fini le tour :
    - on choisit une ville parmi celle non visitées suivant les modalités vues précédemment
    - on rajoute la ville à la liste chemin et on la supprime de la liste des villes restantes.

  On oubliera pas de « *fermer* » le chemin ensuite.

Entrée [4]:

```
def tour(Phi,D):
    """
    Entrées : --> Phi : Matrices donnant la quantité de phéromones ( 0<=Phi(i,j)<=n-1
              --> D : Matrices des poids entre les villes ( 0<=D(i,j))<=n-1 )

    Sortie : villeSuivante : Prochaine ville visitée par la fourmi
    """
    nonVues = [i for i in range (n)]   # Création d'un liste par compréhension des vi
    villeDepart=rd.choice(nonVues)   # Choix d'une ville au hasard
    nonVues.remove(villeDepart)      # on enlève la ville où l'on est au départ
    chemin=[villeDepart]             # On ne mets que la ville de départ
    ville=villeDepart

    while nonVues!=[] : #Tant que l'on a pas exploré toutes les villes

        ville=prochaineVille(ville,nonVues,Phi,D)
        chemin.append(ville)
        nonVues.remove(ville)
    chemin.append(villeDepart)
    return chemin

# Test fonction
#print(tour(Phi,D))
```

Gestion des phéromones. Il s'agit ici de gérer l'évolution de la matrice Φ, plutôt que de faire les calculs coefficients par coefficients, on se propose de les faire matriciellement. Pour cela on a besoin des fonctions suivantes :

- `poids(cycle, D)` qui étant donné un cycle donné sous la forme d'une liste de sommets, renvoie le poids ce celui–ci.

Entrée [5]:

```
def poids(cycle,D):
    """
    Entrées : --> cycle : Liste de sommets des villes
              --> D : Matrices des poids entre les villes ( 0<=D(i,j))<=n-1 )

    Sortie :  --> poidsSommet  : i
    """
    poidsSommet=0  #Initialisation
    for i in range(len(cycle)-1):
        poidsSommet += D[cycle[i]][cycle[i+1]]
    return poidsSommet

# Test
print(poids([1, 3, 4, 0, 2, 1],D))
```

```
2721.0192356566126
```

- `matriceDeChemin(cycle,D)` qui étant donné un cycle `cycle` renvoie la matrice $M = (m_{i,j})_{0 \leqslant i,j \leqslant n-1}$ définie comme suit :

$$m_{i,j} = \begin{cases} \frac{Q}{\pi(\text{cycle})} & \text{si l'arête } \{i,j\} \text{ est parcourue} \\ 0 & \text{sinon.} \end{cases}$$

Entrée [6]:

```
def matriceDeChemin(cycle,D):
    """
    Entrées : --> cycle : Liste de sommets des villes
              --> D : Matrices des poids entre les villes ( 0<=D(i,j))<=n-1 )

    Sortie :  --> M: Matrice m(i,j) : 0<=i,j<=n-1
    """
    M=np.zeros((n,n)) # on initailise une matrice contenant que des 0 sur numpy c'est
    m=Q/poids(cycle,D) # Formule ci-dessus
    for i in range(len(cycle)-1):
        M[cycle[i]][cycle[i+1]]=m
    return M
# Test fonction
#print(matriceDeChemin([0, 1, 3, 2, 4, 0],D))

```

Programmation finale et test

On peut dès lors finir d'implemnter l'algorithme et programmer une fonction `algoFourmis(D, nbIterations)` qui étant donné une matrice des distances (ou poids) `D` et un entier `nbIterations` donnant le nombre d'itération souhaité renvoie la liste des différents poids des parcours des fourmis au cours des itérations. On pourra ainsi visualiser la progression de ces poids en fil des itérations.

Entrée [10]:

```
def algoFourmis(Phi,D,nbIterations):
    progression=[]
    for i in range(nbIterations):
        tourEnCours=tour(Phi,D)
        progression.append(poids(tourEnCours,D))
        Phi=(1-rho)*np.array(Phi) + np.array(matriceDeChemin(tourEnCours,D))
    return progression

resultat = np.around(np.array(algoFourmis(Phi,D,1000)))
print(resultat)

print('le chemin de poids le plus faible est :',min(resultat))
```

```
[11028.  8437. 10757.  9114.  9568.  9721.  9762.  8655. 10128.  9286.
 10656.  9770.  9229.  7939. 10853. 10096.  8761.  8095. 10231. 10082.
  9359.  9798.  8999.  8826. 10398.  8058.  8524. 10335. 10177.  8880.
 10255.  8678. 10412. 10249. 10326.  9029.  8719.  9764.  7574.  9569.
 11065.  8412.  7707. 10370.  9764.  9720.  9040.  9646. 10301.  8389.
 10052. 10319. 10320. 10290.  9104. 10561.  8956.  7980. 10372.  8525.
  8577.  9074.  7280.  9741.  8108.  9604.  8184. 10058.  9000.  9834.
  9518.  9891.  9292.  7959.  7762.  7434.  8451.  9360.  8316.  7312.
  8726.  8651.  8819.  9792.  6979.  9447.  8259.  6771.  8223.  7304.
  6860.  8561.  7319.  7081.  7312.  6779.  9364. 10049.  7744.  6818.
  7081.  6770.  6770.  6965.  7719.  7319.  6965.  8128.  6771.  6812.
  6779.  7294.  7744.  7319.  7252.  7022.  8286.  8246.  7736.  7951.
  8383.  7888.  7022.  7022.  8190.  6901.  8128.  6901.  7114.  8128.
  7121.  7252.  8179.  8522.  6901.  6922.  6901.  7121.  8055.  7319.
  7452.  6965.  8383.  7452.  7319.  6901.  8027.  8288.  7452.  7022.
  8000.  7252.  7121.  8128.  7252.  6901.  6804.  7946.  6929.  8263.
  7175.  7121.  7319.  6901.  8288.  7252.  8522.  6818.  8522.  6901.
  8027.  7022.  8263.  8263.  8190.  6901.  6901.  7452.  8128.  7022.
  8190.  6965.  7252.  8286.  6926.  7605.  7946.  7252.  6965.  8383.
  6965.  8288.  7114.  8146.  8190.  6922.  8522.  6804.  8000.  7175.
  7252.  6922.  7951.  7082.  8179.  6922.  7114.  7319.  7114.  8288.
  6929.  8286.  7114.  8190.  8128.  8288.  6922.  8288.  7022.  7252.
  6818.  6929.  6929.  7114.  7082.  8190.  6901.  6994.  6929.  7252.
  7240.  8190.  6804.  7175.  6929.  7105.  7240.  8190.  8286.  8055.
  6929.  8190.  7319.  7252.  8000.  8288.  7452.  7951.  6922.  7304.
  7304.  8523.  8179.  7175.  7105.  6929.  6926.  8263.  7304.  6926.
  6926.  8383.  6991.  6818.  8190.  7175.  7526.  8128.  7579.  7951.
  6926.  7252.  8055.  8263.  6929.  8522.  6804.  7946.  8263.  8263.
  7082.  6804.  7579.  7951.  8128.  6804.  6818.  7082.  8246.  6818.
  8523.  7304.  8263.  7452.  7022.  8288.  7946.  7105.  7252.  8383.
  8383.  8288.  7082.  6991.  7082.  8383.  8179.  8383.  8000.  7277.
  6937.  8286.  8179.  6818.  8246.  7175.  8179.  6828.  7304.  7951.
  8286.  7252.  8246.  7114.  7951.  8523.  8522.  8128.  6991.  8383.
  7304.  8000.  7114.  6991.  7240.  7082.  8523.  7252.  8179.  8523.
  8000.  7294.  8383.  7175.  6994.  6994.  8190.  7304.  7319.  8055.
  6991.  8179.  8383.  7082.  8523.  7062.  7062.  7579.  8383.  7449.
  8000.  7287.  8288.  8128.  8288.  8523.  8190.  8000.  7252.  7114.
  8288.  6818.  8128.  7022.  8367.  6818.  7297.  7304.  8246.  6828.
  7304.  7605.  8190.  8179.  7951.  7297.  8367.  7304.  8522.  7297.
  7082.  7022.  8190.  7297.  6858.  8179.  6818.  6881.  8522.  8055.
  7319.  8525.  8190.  6858.  8289.  7105.  8000.  7319.  8525.  8179.
```

```
   7452.  8190.  7175.  7022.  7490.  8525.  7069.  7297.  7240.  7114.
   7304.  7297.  7951.  8263.  7175.  8179.  6818.  6818.  7951.  7319.
   7175.  7169.  8522.  8246.  7319.  8128.  7605.  7304.  8128.  7145.
   6829.  7304.  6829.  6818.  6829.  6829.  8522.  6818.  7069.  7069.
   7526.  7022.  8190.  7069.  7605.  7946.  8190.  8525.  7114.  6829.
   7951.  7526.  6994.  8522.  8190.  7526.  6965.  7297.  6881.  7114.
   6849.  7105.  7605.  8000.  6829.  6849.  8288.  8522.  8190.  6849.
   8367.  8128.  7240.  8190.  7452.  8525.  7319.  8190.  7240.  7526.
   7946.  7452.  6965.  6849.  7304.  7297.  7069.  8522.  8288.  8289.
   7169.  6849.  7297.  7169.  7319.  7319.  8263.  7169.  6858.  8367.
   7069.  8367.  7526.  7069.  6860.  7022.  7069.  7319.  6945.  6860.
   7579.  7951.  6849.  8263.  8522.  7297.  8263.  8367.  6849.  7319.
   7452.  7452.  8288.  8375.  7951.  7069.  8522.  8525.  6965.  7380.
   6945.  7105.  6945.  6849.  6945.  8179.  7946.  7297.  8525.  8375.
   7605.  8289.  6858.  7951.  8367.  8375.  7452.  6945.  8522.  8246.
   8375.  7452.  7605.  7297.  8288.  8375.  7452.  7605.  7240.  7297.
   8375.  8522.  7235.  8367.  7951.  7297.  8367.  8128.  6849.  6829.
   7069.  8179.  8522.  8230.  7319.  7169.  6945.  7069.  7579.  7579.
   6849.  6829.  8246.  8375.  6945.  7380.  7452.  7579.  6828.  6851.
   6828.  8289.  6945.  7452.  7114.  7605.  8000.  7105.  6828.  8288.
   7105.  7951.  6828.  8128.  8367.  7114.  6945.  8179.  7951.  8230.
   8000.  7114.  7169.  7526.  7084.  7169.  7069.  8289.  8128.  6849.
   8288.  8288.  7169.  6858.  7951.  7452.  7951.  8179.  7946.  8263.
   8289.  7235.  7022.  6858.  7452.  8522.  7082.  8525.  8288.  7297.
   6860.  6860.  8263.  8179.  6858.  7169.  7526.  8128.  7169.  7169.
   6828.  8230.  8367.  8179.  8179.  8128.  6881.  6881.  7022.  8263.
   6881.  7169.  6965.  7022.  8179.  8522.  7579.  6945.  6860.  8367.
   7169.  8367.  8263.  6945.  8289.  7380.  8525.  8367.  7297.  8263.
   7235.  8000.  6945.  7069.  7579.  7526.  7946.  8179.  8179.  6945.
   8263.  7319.  6945.  8522.  8525.  7319.  7319.  7069.  8375.  8375.
   8288.  7022.  8375.  6860.  8128.  7114.  7114.  7084.  7070.  7082.
   7297.  7526.  7169.  7082.  7235.  8230.  8367.  7579.  8525.  7946.
   8000.  6860.  8179.  6849.  6860.  8246.  7082.  7605.  7070.  8263.
   6860.  6860.  8522.  7946.  8128.  8230.  7605.  7070.  8522.  8375.
   8288.  7605.  7235.  7235.  8375.  7380.  8525.  7069.  6881.  8288.
   7579.  6858.  7114.  7946.  7579.  8525.  8000.  6945.  7235.  7070.
   7070.  6945.  8375.  7319.  8230.  7069.  6858.  7319.  8000.  6860.
   7070.  7114.  8525.  8525.  7082.  8000.  6945.  8525.  7526.  7380.
   8230.  7380.  6945.  8525.  7114.  6945.  7169.  8289.  6860.  8525.
   8246.  8522.  7380.  7084.  7946.  8000.  7946.  7380.  8522.  8000.
   7297.  7452.  8263.  8263.  7319.  7058.  8000.  7319.  7951.  7058.
   7159.  7169.  8522.  8000.  6945.  7235.  8246.  6860.  7235.  7069.
   6858.  6945.  6860.  8179.  8525.  7452.  7069.  6881.  7084.  7169.
   7069.  6828.  8367.  8525.  6945.  6860.  8288.  8179.  6881.  7319.
   7082.  6945.  7069.  7526.  6945.  7058.  6849.  6881.  8367.  7452.
   6945.  6860.  7526.  7452.  6965.  8230.  7084.  7526.  7452.  7069.
   6858.  7526.  8246.  7951.  7319.  7526.  7070.  8288.  7055.  6849.
   8525.  7380.  7082.  8263.  7380.  8375.  6881.  8179.  7084.  6849.
   7951.  6828.  7605.  6945.  7526.  8367.  7069.  7235.  7605.  7380.
   7605.  7319.  7169.  7951.  7069.  6860.  8375.  7084.  7055.  7022.
   7297.  7526.  7069.  6881.  6881.  6828.  7235.  6858.  7055.  8263.
   7084.  7297.  8522.  8246.  6881.  8367.  6945.  7319.  7946.  8375.
   6881.  7169.  7380.  8179.  8289.  6860.  6858.  7452.  6860.  7946.
   7452.  6858.  7069.  6860.  8288.  7605.  7605.  6828.  7169.  8128.
   6860.  7380.  8367.  7297.  7946.  8288.  7452.  8128.  6828.  7526.
   7946.  7082.  8263.  7452.  8375.  6828.  6945.  7526.  8289.  7235.
   7169.  7579.  7082.  6818.  7380.  8375.  7084.  6945.  6818.  8246.
   6945.  7055.  7297.  8289.  7380.  6965.  8000.  8128.  8367.  7082.
   8522.  7084.  7082.  7169.  6818.  8289.  8179.  7082.  7235.  7297.]
le chemin de poids le plus faible est : 6770.0
```

Entrée [ ]:

```
1
```

Entrée [ ]:

```
1
```