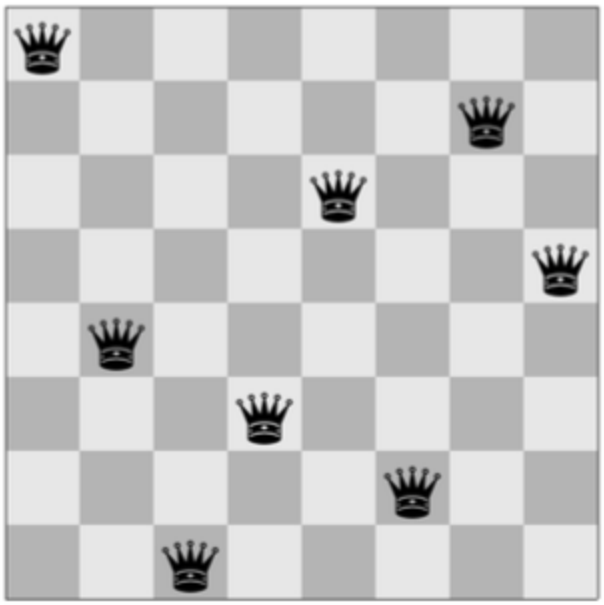
Python for DataScience TD4

Partie 1: Le problème des 8 reines



Le but est de réussir à placer 8 reines sur un échiquier de 8 cases sur 8 cases de telle façon à ce qu'aucune reine ne puisse en attaquer directement une autre. Une reine est attaquée si sur sa ligne, sa colonne ou ses diagonales une autre reine est présente.

Classe individu:

Pour chacune des huit colonnes, on doit reporter la ligne où se trouve la reine. La configuration de reines de l'échiquier ci-dessus s'écrit alors: [0, 4, 7, 5, 2, 6, 1, 3]

Définir la classe individu, ayant comme attributs : une liste de 8 valeurs (ou plus précisément echec_dim valeurs selon la dimension de l'échiquier) et un entier nbconflict représentant le nombre de conflit associé à cet individu (ce n'est pas un attribut à affecter par l'utilisateur au moment de l'instanciation mais à calculer selon la fonction fitness à définir ultérieurement).

Astuce, constructeur avec une liste en paramètre, sans ce paramètre ca génère aléatoirement une liste

```python
import random
echec_dim=8

class individu:
    def __init__(self,val=None):
        if val==None:
            self.val=random.sample(.....................)
        else:
            self.val=val
        #self.nbconflict=self.fitness()#à definier ultérieurment

```

Dans la classe individu, **surcharger une fonction spéciale** de façon à pouvoir écrire print(ind) #ind est une instance de individu et qui imprime l'individu. Exemple : 0 4 7 5 2 6 1 3

Définir la méthode conflict qui retourne true si la reine à la position p1 est en conflit avec la reine en position p2

```python
        def conflict(p1,p2):
        """ retourne true si la reine à la position p1 est
        en conflit avec la reine en position p2"""
        return ......................
```

p1 et p2 deux séquences à deux éléments (position sur l'échiquier), exemple p1=[0,1] p2=[1,2] individu.conflict(p1,p2) doit retourner True.
Définir la méthode fitness qui permet de retourner le nombre de conflit d'un individu

```python
    def fitness(self):
        """ evaluer l'individu c'est connaitre le nobre de conflit"""
        self.nbconflict=0
        for i in .........:
            for j in .....):
                if(individu.conflict ([i,.....],[j,....])):
                    self.nbconflict=self.nbconflict+1
        return self.nbconflict
```

Population

Définir une méthode create_rand_pop(count) qui génère une liste de "count" individus.

Attention : il faut définir cette méthode hors la classe. il ne s'agit pas ni d'une méthode de classe ni d'instance.

Evaluation

Définir une méthode evaluate(pop) qui évalue la population, en gros retourne une liste des individus triés selon le nombre de conflit de chacun.
Astuce: sorted, lambda (faites votre recherche)

Selection

Définir une méthode selection(pop, hcount, lcount) qui retourne une sous population avec les “hcount” premiers éléments et les “lcount” derniers éléments de la liste pop.

Croisement (déjà vue sur nowledgeable)

Définir une méthode croisement(ind1,ind2) qui retourne une liste de deux individus à partir de deux individus ind1 et ind2 (4 premières données de ind1 suivies des 4 dernières de ind2 puis 4 premières données de ind2 suivies des 4 dernières de ind1)

Mutation (déjà vue sur nowledgeable)

Définir une méthode mutation(ind) qui retourne un individu suite à la mutation de ind. Il s’agit de prendre un indice aléatoire de l’individu et de remplacer la donnée correspondante par une nouvelle valeur aléatoire (entre 0 et 7).

Boucle finale

Maintenant que vous avez défini la création d’une population, l’évaluation, la sélection, le croisement et la mutation, il faut mettre en place la boucle permettant de trouver une solution à notre problème.

```
def algoloopSimple():
    pop=create_rand_pop(25) #je commence par créer une population aléatoir de 25 individus
    solutiontrouvee=False
    nbriteration=0
    while not solutiontrouvee: #j'entre dans une boucle jusqu'à ce que je tombe sur une solution
        print("iteration numéro : ", nbriteration)
        nbriteration+=1
        evaluation=evaluate(pop) #j'éavlue la population, le retour est une liste triée selon le nbre de conflit
        if evaluation[0].fitness()==0: # c'est à dire j'ai une solution
            solutiontrouvee=True
        else:                          #j'ai pas de solution
            select=selection(evaluation,10,4) # je selctionne les 10 meilleurs et les 4 pire
            croises=[]
            for i in range (0,len(select),2 ): # je fais le croisement deux par deux
                croises+=croisement(select[i],select[i+1])
            mutes=[]
            for i in select: #j'opère la mutation sur chacun des selctionnés
                mutes.append(mutation(i))
            newalea=create_rand_pop(5) # j'ajoute 5 nouveaux individu aleatoire
            pop=select[:]+croises[:]+mutes[:]+newalea[:] #je recrée ma population :la selection,la mutaion,le croisement et les nouveaux
    print(evaluation[0])
```

En gros, on crée une population aléatoire d’un certain nombre d’individus, on l’évalue, si on on tombe sur un individu à fitness nulle, tant mieux c’est la solution, sinon on sélectionne 10 meilleurs et 4 pires, on croise la sélection deux par deux, on mute chaque individu de la sélection et on crée 5 nouveaux individus, on regroupe le tout pour former la nouvelle population et la boucle est bouclée.

Discussion et échange:

À chaque fois que vous lancez algoloopSimple, est ce que c'est le même nombre d'itération est nécessaire pour trouver une solution?
Si on veut appliquer un algorithme de force brute, quel est l'espace de recherche?
Pourquoi nous ne contentant pas de sélctionner les meilleurs? À quoi sert newalea?

Optionnel : Boucle finale, Toutes les solutions possible

Réécrire le code précédent de façon à récupérer toutes les solutions possibles, commencer par initialiser une liste vide allsolutions, chaque fois que vous tomber sur une solution vous l'ajoutez à la cette liste (si elle n'existe pas déjà) et vous la supprimer de la liste evaluation.
Faites une boucle infinie et imprimez à chaque itération le nombre de solutions atteintes.

Partie 2: Cadre applicatif numpy

Avant de commencer cette partie, supprimez toutes les variables de votre console ou bien démarrez une nouvelle console.

1-A l'aide de numpy créer un tableau avec 500 lignes et 3 colonnes contenant des nombres aléatoires distribués selon une loi normale centrée réduite. Stocker le résultat dans une variable nommée **data**

Observer vos données dans le "Variable explorer" de votre IDE

2-Afficher data dans un scatter plot 3D avec le module Axes3D de mpl_toolkits.mplot3d.

la première colonne sera l'axe des x,la seconde l'axe des y,la troisième l'axe des z

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(10,10))
# choose projection 3d for creating a 3d graph
axis = fig.add_subplot(111, projection='3d')
axis.scatter(
    data[:,0],
    data[:,1],
    data[:,2],

    cmap='plasma'
```

```
)
```

3-Calculer la moyenne de la colonne 0 puis la moyenne de chaque colonne (mean)

3-Calculer l'écart-type de la colonne 0 puis la moyenne de chaque colonne (std)

4- Normalisez data (c'est à dire chaque data[i,j] il faut lui soustraire la moyenne de la colonne et diviser par l'ecart-type de la colonne

```

```

4- Créer la méthode normalisation(data) qui normalise data (synthèse des questions précédentes)

```

```

```

```

5- Appliquer normalisation sur data et mettez le résultat dans **datanorm**

6- calculer la transposé de datanorm

7- calculer le produit matriciel de la transposé de datanorm par datanorm, mettez le résultat dans une variable **covariance**, observer le shape, obtenez le même résultat avec np.cov

8-calculer les valeurs propres et les vecteurs propres de la matrice **covariance**

9-Triez les vecteurs propres en fonction de l'ordre décroissant des valeurs propres (c'est à dire les colonnes de la matrice vecteurspropres

10-créez une matrice **pca** contenant les deux premières colonnes de la matrice de vecteurs propres triés

11- creez projecteddata resulatant du produit datanorm par pca

12-tracez projecteddata

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
fig = plt.figure(figsize=(10,10))
plt.scatter(
    projecteddata[:,0],
    projecteddata[:,1]
    cmap='plasma'
)
```

Discussion