**1 : L'histogramme attendu d'une paire**

Etant donné une paire de cartes privée (celle que je touche au début)

on simule n (cible 1000) fois 5 cartes visibles (celles du milieu)

            au sein de chacune de ces n simulations on simule m (cible 1000) simulations de 2 cartes (représentant les 2 cartes cachées de l'adversaire).

           pour chacune des m simulations on détermine qui gagne. Une fois les m simulations faites on calcule la probabilité qu'on a de gagner (nb de fois où on a gagné sur les m simulations /m).

On obtient donc pour chacune des n simulations une probabilité de gagner.

On regrouper ces informations dans un histogramme en 10 parties.

partie 1 (bin 1) : abscisse =1 , hauteur de la barre = nb de simulations parmi les n ou proba de gagner <0.1

partie 2 (bin 2) : abscisse =2 , hauteur de la barre = nb de simulations parmi les n ou proba de gagner dans [0.1,0.2[

etc  
partie 10 (bin 10) : abscisse =10 , hauteur de la barre = nb de simulations parmi les n ou proba de gagner =>0.9

**2 : calcul de distance entre 2 histogrammes : earth's mover distance**

Voici une solution simple pour calculer le earth's mover distance.

1 : cette mesure est également appelée la distance de wasserstein et elle est implémentée dans scipy.

2 : voici un exemple simple d'usage infra : en voici les explications :

- tu dois définir les bins, ie les coordonnées des "tas" (rappel la distance de wasserstein mesure l'effort à faire pour passer d'une distribution de "tas" à une autre, cet effort dépend de la quantité à déplacer mais également de la distance de déplacement). Pour les histogrammes des distributions pour le projet ce sera [1,2,3,4,5,6,7,8,9,10] (un découpage en 10 me semble suffisant).

- pour chaque bin tu donnes le pourcentage de simulations qui sont dans ce bin (donc la somme de ces pourcentages sur l'ensemble des bins vaut 100 ou 1 : cela ne change rien c'est juste un facteur d'échelle). Mettons que la somme vaille 1.

- dans les exemples infra :

1er cas : il faut déplacer 0,6 d'un bin à l'autre (0.2+0.6=0.8).

2e cas :  sous cas 1 : il faut déplacer 1 de un bin soit 1\*1 (car l'espace entre bins est de 1) ; sous cas 2 : il faut déplacer 1 de 2 bins soit 1\*2.

from scipy import stats

bin\_locations = [0, 1]

print(stats.wasserstein\_distance(bin\_locations, bin\_locations, [.2, .8], [.8, .2]))

bin\_locations = [0,1,2]

print(stats.wasserstein\_distance(bin\_locations, bin\_locations, [0,0,1], [0,1,0]))

print(stats.wasserstein\_distance(bin\_locations, bin\_locations, [0,0,1], [1,0,0]))

**3 : CAH Python**

Lorsqu’on fait une CAH on peut avoir 2 types de données en entrée :

Cas 1 : une table avec les observations en ligne et les variables en colonne. C’est le cas le plus fréquent.

Cas 2 : on a une matrice n\*n des distances entre observations (n étant le nombre d’observations).  **C’est le cas ici.**

Voici une 1re solution pour faire une CAH dans le cas 2 en python <https://stackoverflow.com/questions/47321133/sklearn-hierarchical-agglomerative-clustering-using-similarity-matrix>

Mais ce qui m’embête c’est qu’on ne peut pas appliquer le critère de Ward dans ce cas-là (or c’est le plus souvent utilisé).

2e solution : demande un peu plus de travail mais permet d’utiliser Ward avec scipy

from scipy.cluster.hierarchy import dendrogram,linkage,fcluster

import scipy.spatial.distance as ssd

from matplotlib import pyplot as plt

distArray = ssd.squareform(m)  # m est ta matric de distance n\*n, cette ligne ne garde que la moitié de la matrice au-dessus de la diagonale (toute l’info est dedans)

Z = linkage(distArray, 'ward')

# calculate full dendrogram

plt.figure(figsize=(25, 10))

plt.title('Hierarchical Clustering Dendrogram')

plt.xlabel('sample index')

plt.ylabel('distance')

dendrogram(

    Z,

    leaf\_rotation=90.,  # rotates the x axis labels

    leaf\_font\_size=8.,  # font size for the x axis labels

)

plt.show()

max\_d = 3

res=clusters = fcluster(Z, max\_d, criterion='distance') # tu dois regader les distances dans le dendogramme pour ensuite fixer un seuil ici on calcule les clusters finaux

print(res)