Expérience 1 :

Le tableau ci-après présente les résultats de l'expérience.

Points	Point1	Point2	Point3	Point4	Point5	Point6
Nombre de voisins méthode linéaire	5	2	1	17	2	2
Nombre de voisins avec k-d tree	5	2	1	17	2	2

Pour chacun des points, le nombre de voisins est le même, peu importe la méthode utilisée. Cependant, dans les fichiers .text en fonction de la méthode utilisée pour trouver les voisins, l'ordre des voisins trouvés pour chaque point sera différent dans les fichiers pour les voisins obtenus avec la méthode linéaire et dans ceux obtenus en utilisant un arbre k-d tree.

Pour pouvoir s'assurer que le nombre de voisins d'un point est le même peu importe la méthode, on a juste imprimé à l'écran la taille des listes contenant les résultats de différentes méthodes rangeQuery et on a comparé les points contenus dans les fichiers .text.

Expérience 2 :

Le tableau ci-après présente les résultats de l'expérience.

Noms des fichiers de points	Temps de calcul moyen pour la méthode linéaire (en ms)	Temps de calcul moyen pour la méthode utilisant l'arbre k-d tree (en ms)
Point_Cloud_1.csv	0	0
Point_Cloud_2.csv	2	0
Point_Cloud_3.csv	5	1

J'ai calculé le temps moyen qu'il faut pour trouver les voisins des points aux positions multiples de 10 dans les différents fichiers de nuages de points en utilisant l'arbre k-d tree et la méthode linéaire avec un eps égale à 0.5. De ce qu'on observe dans le tableau, le temps de calcul moyen pour trouver les voisins d'un point dans un nuage de points en utilisant un arbre k-d tree est plus rapide que celui utilisant une méthode linéaire. Ce résultat traduit parfaitement ce à quoi on s'attendait.

Expérience 3 :

Le tableau ci-après présente les résultats de l'expérience.

Noms des fichiers de points	Temps d'exécution du DBScan avec la classe NearestNeighbors (en ms)	Temps d'exécution du DBScan avec la classe NearestNeighborsKD (en ms)
Point_Cloud_1.csv	645	616
Point_Cloud_2.csv	836	878
Point_Cloud_3.csv	845	626

On observe qu'en fonction du fichier de points, le temps d'exécution de DBScan avec les deux méthodes différentes donnent des résultats distincts. Pour les fichiers Point_Cloud_1.csv et Point_Cloud_3.csv, le temps d'exécution de DBScan utilisant la classe NearestNeighborsKD est meilleur que celui de DBScan utilisant la classe NearestNeighbors alors que pour le fichier Point_Cloud_2.csv, celui du DBScan utilisant la classe NearestNeighbors qui est le meilleur. Néanmoins, au fur et à mesure qu'on répète l'expérience, on se rend compte que le temps d'exécution du DBScan est quasiment le même avec les 2 méthodes.

Les résultats obtenus ne sont pas vraiment ceux auxquels je m'attendais. Je pensais que l'exécution du DBScan en utilisant la classe NearestNeighborsKD serait beaucoup plus rapide.