Rapport Projet Python

I – Prérequis :

Afin que mon programme approximate_pi.py puisse fonctionner il est nécessaire d'installer le package python OpenCV (pip install opencv-python) car j'utilise le module cv2 de celui-ci.

II – Simulator.py :

Le cœur de **simulator.py** est la fonction **simulateur()** dont la complexité est globalement linéaire par rapport au nombre de points simulés.
Premièrement car elle va créer une liste de points aléatoires de la taille **nb_points_a_simuler** avec la fonction **genere_points_aléatoire()**.
Deuxièmement car pour chaque point elle va calculer avec **estDansCercle()** si le point appartient au cercle de centre (0,0) et de rayon 1.

III – Approximate_pi.py :

1. Le poids des images :

- Au départ ne connaissant pas bien le format d'image PPM j'ai développé ma fonction **generate_ppm_file** en écrivant au format **P3** cependant avec celui-ci les images étaient lourdes. Afin d'obtenir des images compressé 2x plus légère en moyenne je me suis donc tourné vers le format **P6** qui code les images en binaire.

- En effet le poids des images est lié seulement au paramètre **taille_image** et ne dépend pas du nombre de points simulés car pour construire l'image ppm on écrit dans un fichier pour chaque pixel la couleur voulu à l'aide d'un tuple. Le reste des informations contenu dans le fichier est négligeable comparé à la taille de l'image donc le poids de l'image est linéaire à la taille de l'image.

2. Le problème de l'écriture du nombre π sur l'image PPM généré:

Dans un 1er temps afin d'afficher le nombre π sur l'image j'ai codé une fonction qui modifiait la liste représentant l'image. Cependant cela me forçait à chaque fois qu'un dixième de point était tiré de réécrire intégralement la liste représentant l'image pour y afficher le nombre π. Afin de diminuer l'espace mémoire nécessaire j'ai décidé d'utiliser module **cv2** du package **Opencv**, de cette façon à chaque fois qu'un dixième des points sont tirés je peux les rajouter sur la même liste, créer l'image et enfin ouvrir cette image pour y écrire le nombre π.
Je n'ai donc plus besoin de recopier intégralement la liste à chaque nouveau tirage, le coût devient donc linéaire de la taille de l'image.

3. Temps d'exécution :

- Approximativement linéaire du poids de vue du nombre de points simulés.

En effet si l'on fixe le paramètre **taille_image** mais que l'on fait varier le paramètre **nombre_points_simules** pour **Approximate_pi.py** cela va se ressentir à deux endroits au niveau de la fonction **main()**. Le 1er est au début de la fonction lors de l'appel à la fonction **simulateur()** de **simulator.py** pour stocker les points dans des listes, sa complexité est donc linéaire par rapport au nombre de points simulés comme vu précédemment.

Le 2nd est au niveau de **generate_ppm_file()** qui lors du parcours des listes **points_dans_cercle** et **points_hors_cercle** va pour chaque point de ces deux listes modifier l'élément qui lui correspond dans **liste_ppm** qui représente l'image.

Le coût est donc ici également linéaire par rapport au nombre de points simulés mais celui-ci est négligeable par rapport au coût de l'appel à la fonction **simulateur()** L'explication est qu'il n'y a en tout qu'une dizaine d'opérations mathématiques et un remplacement d'élément dans une liste pour chaque point au moment du parcours des listes contre bien plus d'opérations dans **simulateur()**.

- Approximativement linéaire du point de vue de la taille de l'image.

Cela s'explique d'abord car une plus grande taille de l'image signifie une plus grande liste pour la représenter à créer, ce qui prend plus de mémoire.
Dans **genere_ppm_file()** afin de créer l'image ppm on doit parcourir chaque tuple représentant chaque point dans la liste représentant l'image. Augmenter de x la taille de l'image revient donc à augmenter de 3x ce nombre d'éléments à parcourir.
Enfin d'après les tests de performances que j'ai effectués sur le module **cv2** de **OpenCV** écrire un nombre sur une image à l'aide de ce module est linéaire par rapport à la taille de cette image. Or j'utilise ce module à la fin de **genere_ppm_file()** pour afficher le nombre π sur l'image.

- Il est à noter que si l'on fait varier les deux paramètres c'est le nombre de points simulés qui va être négligeable devant la taille de l'image pour le temps d'exécution. Cela s'explique car on parcourt de nombreuses fois la liste représentant l'image mais globalement qu'une seule fois les listes représentant les points simulés.