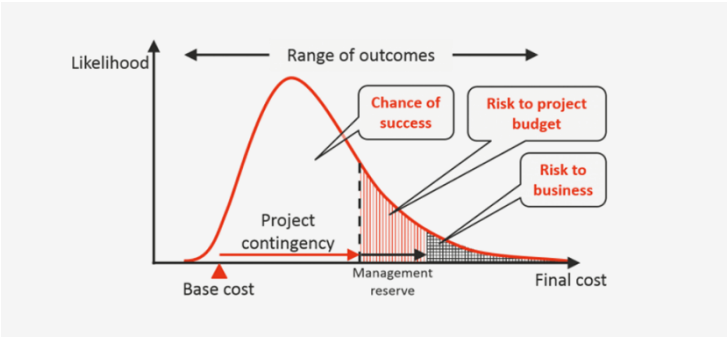
Rapport TP03 Simulation

Lab 3 - Simu PI Et Conf Intervals

Année 2024-2025

Présenté par :
MOTA TITOUAN



Université Clermont-Auvergne

L2 Informatique

18 février 2025

Table des matières

Chapitre 1

Introduction

1.1 Contexte du tp

Ce TP s'inscrit dans le cadre du cours Z****** "Projet informatique", une introduction à la simulation. Ce cours aborde diverses notions, telles que la génération de nombres aléatoires et pseudo-aléatoires, ainsi que des méthodes de simulation comme celle de Monte Carlo. Il sensibilise à l'utilisation de certains générateurs de nombres qui ne sont pas adaptés à la recherche scientifique en raison de leur distribution non uniforme, des corrélations entre les valeurs successives et des cycles courts (période de répétition limitée).

1.2 Méthode de génération des nombres aléatoires

Comme mentionné dans la section 1.1, la plupart des générateurs de nombres inclus dans les langages de programmation ne sont pas adaptés à la recherche scientifique pour diverses raisons. Pour réaliser ce TP, nous avons utilisé la méthode MT19937-64, une implémentation en langage C de la méthode Mersenne Twister proposée par Takuji Nishimura et Makoto Matsumoto [1] pour générer des nombres pseudo-aléatoires. Nous avons choisi cette méthode, présentée dans le LAB 2, car elle offre une période extrêmement longue de $2^{19937} - 1$, permettant de générer une séquence de nombres pseudo-aléatoires sans répétition pendant une très longue durée, ainsi qu'une distribution uniforme des nombres générés. Elle possède également d'autres propriétés intéressantes, comme une faible corrélation entre les valeurs successives, une vitesse de génération rapide et une fiabilité validée par la communauté scientifique.

Pour générer les nombres aléatoires, nous utilisons la fonction définie par :

Listing 1.1 – mt64.h

```
double genrand64_real1(void);
```

Cette fonction génère des nombres réels de type double dans l'intervalle [0,1], ce qui est parfaitement adapté aux questions à venir.

1. Site officiel de Makoto Matsumoto (MT19937-64)

Chapitre 2

Lab 3 Monte Carlo Simulation & Confidence Intervals

2.1 Question 1 - Volume of a sphere with the Monte Carlo method

Dans cette question, nous allons :
- Créer une fonction qui calcule le volume d'une sphère par la méthode de Monte Carlo
- Tester avec différents nombres de points
- Observer la précision selon le nombre de points
- Obtenir aussi une estimation de π.

2.1.1 Fonction qui calcule le volume d'une sphère par Monte Carlo

Nous avons donc implémenté cette fonction en langage C avec la méthode de génération aléatoire MT19937-64 (version du 2004/09/29).

```
double estimateSphereVolumeMonteCarlo(double numIterations) {
        int          pointsInsideSphere = 0;
        double       randomX, randomY, randomZ;
        double       distanceFromOrigin;

        for (int i = 0; i < numIterations; i++)
        {
                randomX = genrand64_real1();
                randomY = genrand64_real1();
                randomZ = genrand64_real1();

            distanceFromOrigin = sqrt(pow(randomX, 2) + pow(
              ↪ randomY, 2) + pow(randomZ, 2));

                if (distanceFromOrigin <= 1)
                {
                        pointsInsideSphere += 1;
```

```
            }
        }

        return ((double)pointsInsideSphere / numIterations) * 8;
}
```

Cette fonction prend en paramètre un double qui représentera le nombre d'itérations souhaité (nombre de simulations). Ensuite, nous avons les déclarations des différentes variables :

- **pointsInsideSphere** : nombre de points qui se situent dans la sphère, autrement dit le nombre de points qui sont à une distance inférieure ou égale à 1 du point (0,0,0)
- **randomX,randomY,randomZ** : coordonnées du point (x, y, z) générées aléatoirement dans l'intervalle [0,1]
- **distanceFromOrigin** : sert à stocker la distance donnée par la formule :

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Par la suite, une boucle allant de 0 à nbIteration-1 démarre et génère à chaque fois une coordonnée aléatoire pour rx, ry, rz. Ensuite, si la distance est plus petite que 1, on incrémente nbPointsInSphere de 1, puis on répète l'opération nbIteration fois. À la fin, on retourne la moyenne des points qui sont dans la sphère (nbPointsInSphere/nbIteration) que nous multiplions par 8 car nous générons des points dans le cube $[0,1]\times[0,1]\times[0,1]$, qui ne représente qu'un octant (1/8) de l'espace total.

2.1.2 Tester avec différents nombres de points

Dans un premier temps, nous savons que le résultat attendu est 4,18879. Pour mieux visualiser les résultats, j'ai utilisé le logiciel ParaView qui est un logiciel de visualisation scientifique. J'ai modifié mon code pour que chaque point généré dans le programme soit inscrit dans un fichier .csv. L'algorithme ressemble maintenant à :

```
double estimateSphereVolumeMonteCarlo(double numIterations) {
        ...

        FILE *fichier = fopen("sphere_points.csv", "w");
        if (fichier == NULL) {
                printf("Erreur lors de l'ouverture du fichier\n"
                    ↪ );
                return -1;
        }

        fprintf(fichier, "\"X\",\"Y\",\"Z\",\"Inside\"\n");


        for (int i = 0; i < nbIteration; i++) {
                ...
                }
```

```
                fprintf(fichier, "%.6f,%.6f,%.6f,%d\n", rx, ry,
                    ↪ rz, isInside);
            }

            fclose(fichier);
            ...
}
```

Un fichier est généré avec les coordonnées X, Y, Z de chaque point ainsi qu'un indice 1 si le point est dans la sphère et 0 sinon.

— **Pour 1000 itérations** : Nous obtenons une moyenne de 4.232000.

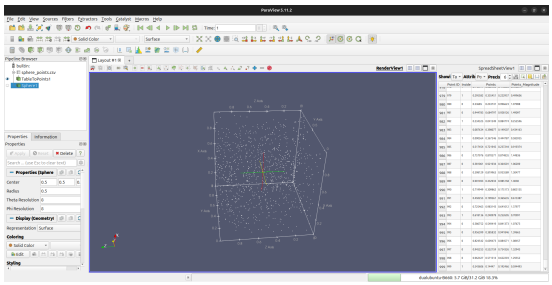


FIGURE 2.1 – Nuage de points généré par le programme (1000)

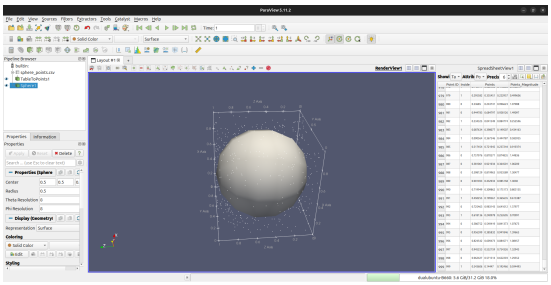


FIGURE 2.2 – Nuage de points généré par le programme + sphère (1000)

— **Pour 1.000.000 itérations** : Nous obtenons une moyenne de 4.187544.

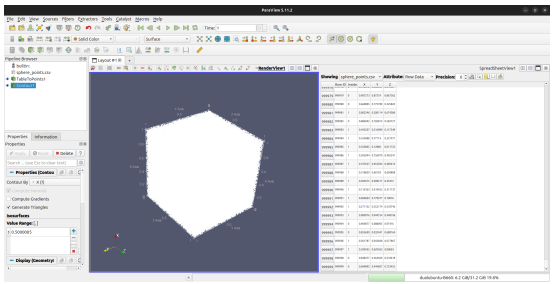


FIGURE 2.3 – Nuage de points généré par le programme (1.000.000)

— **Pour 1.000.000.000 itérations** : Nous obtenons une moyenne de 4.188695.

(Pour une meilleure visibilité, toutes les images sont disponibles à la fin du rapport en grand format.)

2.1.3 Observer la précision selon le nombre de points

Nous constatons que plus nous augmentons le nombre de répétitions, plus le résultat est précis, ce qui était prévisible. Cette observation s'explique par le fait que la méthode de Monte Carlo repose sur des échantillons aléatoires pour estimer une valeur. En augmentant

le nombre de points, nous réduisons l'erreur. En d'autres termes, avec plus de points, la moyenne des résultats converge vers la valeur théorique attendue

2.1.4 Obtenir aussi une estimation de π.

Pour obtenir une estimation de π, il nous suffit de modifier le retour de la fonction en utilisant les informations suivantes :

Le volume V de la sphère est donné par la formule :

$$V = \left(\frac{\text{nbPointsInSphere}}{\text{nbIteration}} \right) \times 8$$

Sachant que le volume d'une sphère est également donné par :

$$V = \frac{4}{3} \pi r^3$$

Nous pouvons résoudre pour π :

$$\pi = \frac{V \times 3}{4 \times r^3}$$

Avec $r = 1$, l'expression se simplifie à :

$$\pi = \frac{V \times 3}{4}$$

Ainsi, le code pour estimer π est le suivant :

```
double volumeSphere = ((double)pointsInsideSphere /
    ↪ numIterations) * 8;
return (volumeSphere * 3) / 4;
```

- **Pour 1000 itérations** : 3.174000
- **Pour 1.000.000 itérations** : 3.140658
- **Pour 1.000.000.000 itérations** : 3.141521

Pour un résultat théorique attendu de : 3.141592

2.1.5 Analyse des résultats

Nous observons que l'estimation de π se rapproche de la valeur théorique attendue à mesure que le nombre d'itérations augmente. Cela illustre bien le principe de la méthode de Monte Carlo, où une plus grande quantité de points permet d'obtenir une meilleure approximation.

Avec 1000 itérations, l'estimation est de 3.174000, ce qui est relativement éloigné de la valeur théorique. En augmentant le nombre d'itérations à 1.000.000, l'estimation s'améliore à 3.140658. Enfin, avec 1.000.000.000 itérations, nous obtenons une estimation très proche de la valeur théorique, soit 3.141521.

Ces résultats montrent l'importance d'utiliser un nombre suffisamment élevé d'itérations pour obtenir des estimations précises avec la méthode de Monte Carlo.

2.2 Question 2 - Computing independent experiments and obtaining the mean

Pour cette question, nous allons créer une fonction qui répète plusieurs fois l'expérience vue dans la section précédente et qui calcule la moyenne des résultats obtenus.

2.2.1 Fonction qui effectue n fois l'expérience

Nous avons implémenté cette fonction en langage C en utilisant la méthode de génération aléatoire MT19937-64 (version du 29/09/2004) et la fonction `estimateSphereVolumeMonteCarlo` vue précédemment.

```
double runMonteCarloSimulations(double numPoints, double
    ↪ numSimulations){
        long double sumOfResults = 0;

        for (int i = 0; i < numSimulations; i++)
        {
                sumOfResults += estimateSphereVolumeMonteCarlo(
                    ↪ numPoints);
        }


        return sumOfResults / numSimulations;
}
```

Cette fonction prend deux paramètres :
— **numPoints** : Le nombre de points générés dans une simulation.
— **numSimulation** : Le nombre de simulations effectuées.
Elle effectue `nbSimu` simulations, additionne chaque résultat à `result`, puis retourne la moyenne des résultats (`result / nbSimu`).

2.2.2 Test avec différentes valeurs

NbPoints / **nbSimulation**	**1000**	**1 000 000**	**1 000 000 000**
10	4.153600	4.189119	4.188819
20	4.148400	4.188577	4.188789
30	4.158400	4.188006	4.188787
40	4.167000	4.187847	4.188799

TABLE 2.1 – Résultats des simulations avec différents nombres de points et de simulations.

Nous cherchons 4.188790.

2.2.3 Précision et nombre de points

L'augmentation du nombre de points utilisés dans les simulations améliore considérablement la précision de l'estimation du volume. Avec 1.000.000.000 points, l'erreur relative

est extrêmement faible, souvent inférieure à 0.001%. Cela montre l'efficacité de la méthode de Monte Carlo pour estimer des volumes lorsque le nombre de points est suffisamment élevé.

2.2.4 Précision et nombre de simulations

Pour un nombre donné de points, l'augmentation du nombre de simulations ne montre pas une amélioration significative de la précision. Cela suggère que l'erreur est principalement due à la méthode de Monte Carlo elle-même et non au nombre de simulations.

2.2.5 Résultats de l'erreur relative

-**Pour 1000 points** : Les erreurs relatives varient entre 0.52% et 0.96%.
-**Pour 1.000.000 points** : Les erreurs relatives varient entre 0.0051% et 0.0225%.
-**Pour 1.000.000.000 points** : Les erreurs relatives sont extrêmement faibles, souvent inférieures à 0.001%.

2.2.6 Conclusion

La précision de l'estimation du volume de la sphère s'améliore significativement avec l'augmentation du nombre de points utilisés dans la simulation. Pour des estimations encore plus précises, il serait bénéfique d'augmenter le nombre de points, bien que cela puisse nécessiter plus de ressources computationnelles.

2.3 Question 3 - Computing of confidence intervals around the simulated mean

2.3.1 Fonction qui calcul l'intervale de confiance

La fonction prend en paramete :
— **numSimulations** : nombre de simulation
— **numPoints** : nombre de points par simulation

```
double * calculateConfidenceInterval(double numSimulations,
    ↪ double numPoints){

        double * confidenceInterval = malloc(2 * sizeof(double))
            ↪ ;
        double *results = malloc(numSimulations * sizeof(double)
            ↪ );
        double moyenne = 0;
        double varianceSum = 0;

        if(confidenceInterval == NULL || results == NULL)
        {
                return NULL;
        }
```

```
        for (int i = 0; i < numSimulations; i++)
        {
                results[i] = estimateSphereVolumeMonteCarlo(
                    ↪ numPoints);
                moyenne += results[i];
        }

        moyenne = moyenne / numSimulations;

        for(int i = 0; i < numSimulations; i++)
        {
                varianceSum += (results[i] - moyenne) * (results
                    ↪ [i] - moyenne);
        }

        double variance = varianceSum / (numSimulations - 1);

        double confidenceRadius = 2.704 * sqrt(variance /
            ↪ numSimulations);

        confidenceInterval[0] = moyenne - confidenceRadius;
        confidenceInterval[1] = moyenne + confidenceRadius;

        free(results);
        return confidenceInterval;
}
```

La fonction crée un tableau de taille n et le remplit avec n simulations de Monte Carlo. Ensuite, elle calcule la moyenne des résultats. Pour chaque valeur, on soustrait la moyenne, puis on élève chaque valeur au carré. Une fois la somme des carrés calculée, nous la divisons par $n-1$, ce qui nous permet d'obtenir la variance.

Pour obtenir l'intervalle de confiance, il nous faut choisir la valeur adéquate dans le tableau de la distribution de Student. Ici, pour un niveau de confiance de 99% et 39 degrés de liberté, la valeur critique est $t = 2.704$.

2.3.2 Test

Pour un intervalle de confiance à 99% et 39 degrés de liberté, nous obtenons un intervalle de $[4.1864, 4.1893]$, ce qui semble cohérent.

Chapitre 3

Conclusion

Dans ce rapport, nous avons exploré la méthode de Monte Carlo pour estimer le volume d'une sphère de rayon 1 et avons évalué la précision des résultats en fonction du nombre de points et de simulations. Nous avons également calculé des intervalles de confiance pour valider nos estimations.

Les résultats montrent que l'augmentation du nombre de points utilisés dans les simulations améliore significativement la précision de l'estimation du volume. Avec un nombre élevé de points, l'erreur relative devient extrêmement faible, ce qui démontre l'efficacité de la méthode de Monte Carlo pour ce type de problème. En revanche, l'augmentation du nombre de simulations n'a pas montré d'amélioration significative de la précision, suggérant que l'erreur est principalement due à la méthode elle-même.

L'intervalle de confiance obtenu pour un niveau de 99% avec 39 degrés de liberté était de [4.1864, 4.1893], ce qui est cohérent avec le volume théorique attendu de 4.18879. Cela valide la fiabilité de notre approche.

En conclusion, la méthode de Monte Carlo est une technique puissante pour estimer des volumes dans des espaces de grande dimension, et son efficacité peut être améliorée en augmentant le nombre de points utilisés dans les simulations. Pour des résultats encore plus précis, il serait bénéfique d'augmenter le nombre de points, bien que cela puisse nécessiter plus de ressources.

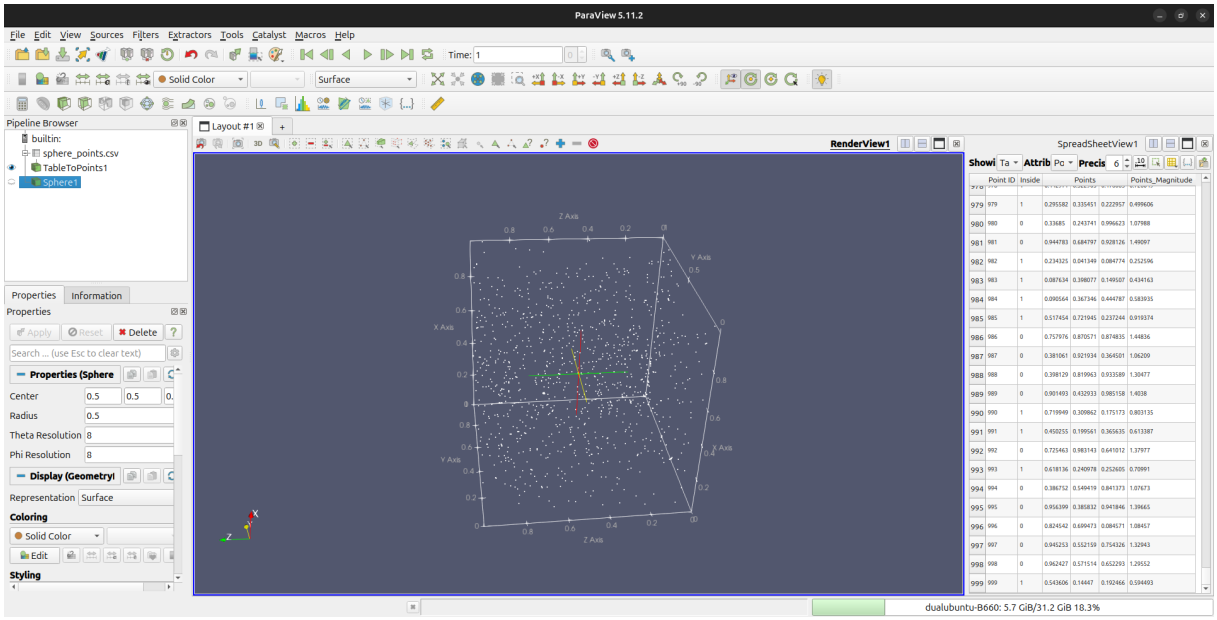


FIGURE 3.1 – Nuage de points généré par le programme (1000)

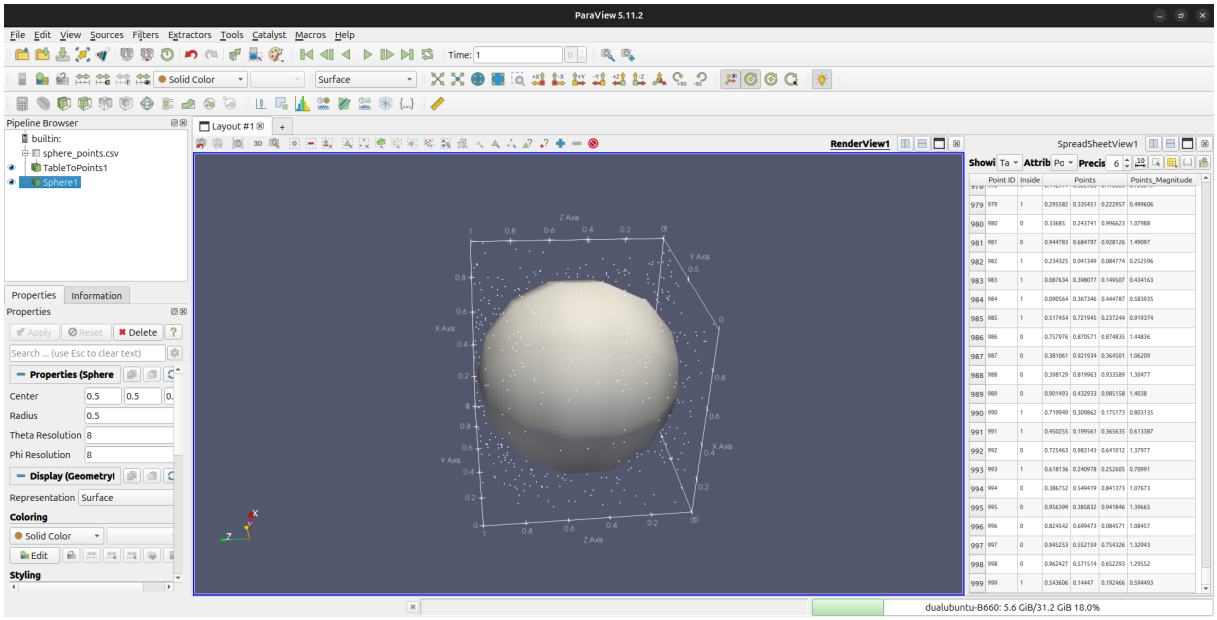


FIGURE 3.2 – Nuage de points généré par le programme + sphère (1000)

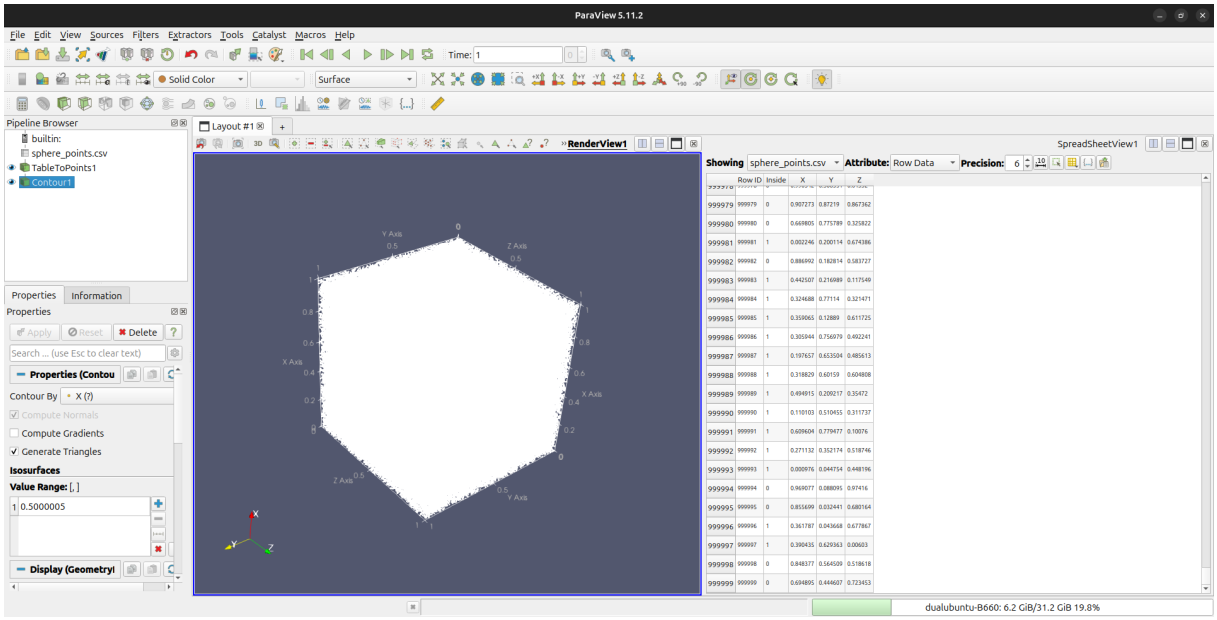


FIGURE 3.3 – Nuage de points généré par le programme (1.000.000)