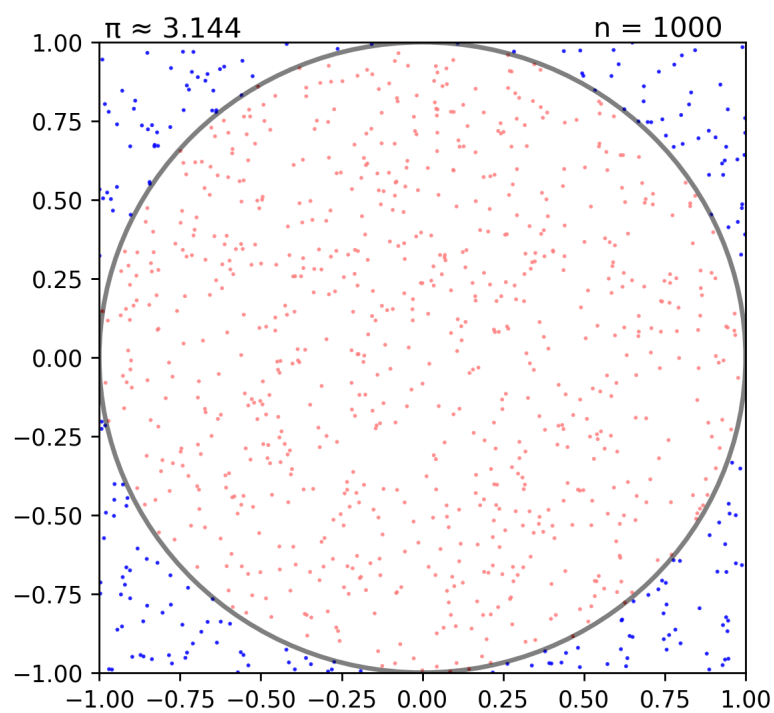


TP

# Monte Carlo Simulation et intervalles de confiance



CHASSAGNOL Rémi  
ZZ2 - F2 - promo24

*Professeur:* HILL Davide

7 OCTOBRE 2022

## Contents

<b>1</b>	<b>Simulation de <math>\pi</math></b>	<b>2</b>
<b>2</b>	<b>Calcul de précision</b>	<b>3</b>
<b>3</b>	<b>Calcul d'intervalles de confiance à 95%</b>	<b>4</b>

# 1 Simulation de $\pi$

Le but de cette première partie est de déterminer une valeur approchée de  $\pi$  en utilisant une méthode de **Monte Carlo**. Le principe est le suivant:

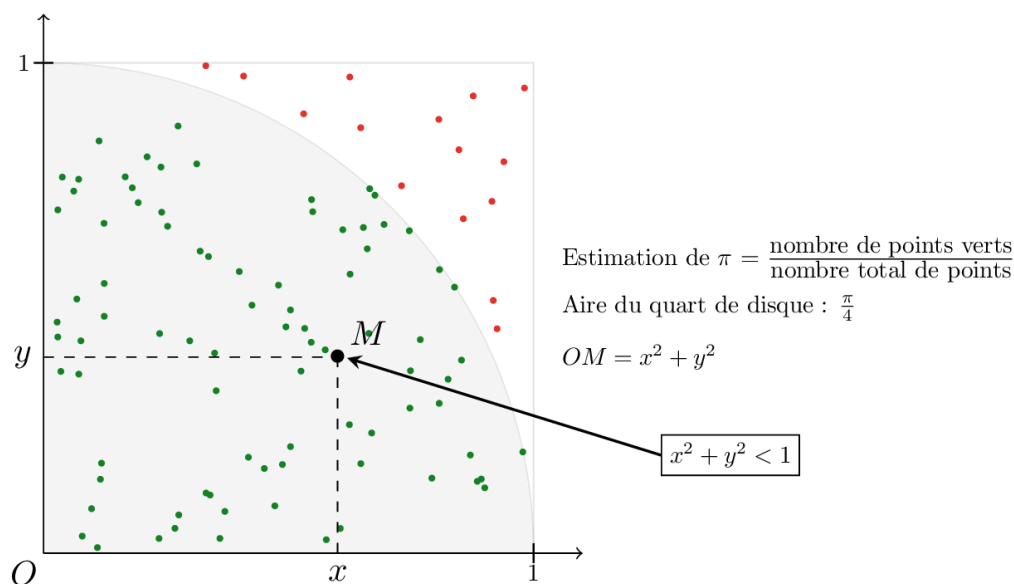


Figure 1: Simulation  $\pi$  Monte Carlo

1

On tire deux nombres aléatoires  $x$  et  $y$ , compris entre 0 et 1, à l'aide de **Mecene Twister**. C'est deux nombres sont les coordonnées d'un point appartenant un à espace représenté par un carré de coté 1.

Ensuite, on teste si  $x^2 + y^2 < 1$ , si c'est le cas, cela signifie que le point trouvé se trouve dans le quart de cercle de rayon 1 situé à l'intérieur de notre carré comme ci-dessus. À la fin on fait le rapport du nombre de points trouvés dans le quart de cercle sur le nombre de points total et on obtient une valeur approchée de  $\frac{\pi}{4}$ . Là il suffit de multiplier par 4 pour trouver  $\pi$ .

<sup>1</sup>Figure 1: [https://media.eduscol.education.fr/ftp\\_eduscol/2019/Ressources/Mathematiques/RA19\\_Lycee\\_G\\_1\\_MATH\\_Algorithmique\\_et\\_Programmation\\_activite\\_11.html](https://media.eduscol.education.fr/ftp_eduscol/2019/Ressources/Mathematiques/RA19_Lycee_G_1_MATH_Algorithmique_et_Programmation_activite_11.html)

Pour l'implémentation, on va utiliser `genrand_real1()` pour avoir des nombres aléatoire dans  $[0; 1]$  et on suit les étapes expliquées précédemment. La fonction prend en paramètre le nombre de points à générer. Comme Mercene Twister est un bon générateur qui explore bien l'espace, plus on génère de points, plus on couvre l'espace et ce de manière "*uniforme*". De ce fait, plus on augmente le nombre de points générés, plus la précision augmente.

```
double simPi(unsigned int nbPoints) {
    unsigned int pointsInCircle = 0;
    unsigned int i;
    double x, y;

    for (i = 0; i < nbPoints; ++i) {
        x = genrand_real1();
        y = genrand_real1();

        if (x * x + y * y < 1) {
            pointsInCircle++;
        }
    }

    return (double)(4 * pointsInCircle) / nbPoints;
}
```

Voici le résultat pour différents nombres de générations:

points générés	valeur de $\pi$ trouvée
1000	3.124000
1000000	3.144720
1000000000	3.141541

On constate bien que plus on génère de points, plus on se rapproche de la valeur de  $\pi$ .

## 2 Calcul de précision

Dans cette deuxième partie, on va réutiliser la fonction `simPi()` implémentée précédemment, et on va s'intéresser à la précision. Le but va être de calculer une moyenne sur un certain nombre de simulation de  $\pi$ . Pour chacune des moyennes, la simulation se fera avec différents nombres de points à générer. Ensuite, on compare la valeur de  $\pi$  trouvée (en moyenne) avec `M_PI` qui est une macro de `math.h` qui contient une valeur relativement précise de  $\pi$ .

Notre fonction `simPiPrecisison()` va donc prendre deux paramètre:

- **nbPoints**: le nombre de points à générer avec `simPi()`
- **nbSimulations**: le nombre d'expériences à effectuer pour le calcul de la moyenne

La fonction affiche simplement le pourcentage de différence entre la valeur de  $\pi$  simulée et `M_PI`.

```
void simPiPrecisison(unsigned int nbPoints, int nbSimulations) {
    int i;
    double mean;
    double sum = 0;

    initMT();
```

```

// calcul des pis
for (i = 0; i < nbSimulations; ++i) {
    sum += simPi(nbPoints);
}
mean = sum / nbSimulations;

printf("mean: %lf\n", mean);
printf("precision: %lf\n", fabs(mean - M_PI) / M_PI);
}

```

Voici les résultats pour des moyennes calculées sur 40 Simulations:

points générés	moyenne	précision
1000	3.142400	0.000257
1000000	3.141186	0.000129
1000000000	3.141586	0.000002

### 3 Calcul d'intervalles de confiance à 95%

Dans cette dernière partie, on va écrire une fonction qui permet de calculer et d'afficher l'intervalle de confiance à 95% pour les moyennes simulées précédemment. Pour ce faire on commence par calculer la moyenne arithmétique  $\bar{X}(n)$  sur un certain nombre de simulations de  $\pi$ . En même temps, on sauvegarde les valeurs de  $\pi$  trouvées dans un tableau. Ensuite on utilise les moyennes et le tableau de valeurs de  $\pi$  pour calculer  $S^2(n) = \frac{\sum_{i=1}^n X_i - \bar{X}(n)}{n-1}$ , qui est un estimateur sans biais de la variance. Enfin on utilise une loi de Student pour calculer le taux d'erreur.