Compte rendu du TP3 de simulation

Mathieu VILLEDIEU DE TORCY

Sommaire :

Introduction

Le but du TP est d'estimer la valeur de *PI* à l'aide de la méthode de *Monte Carlo* et d'ensuite calculer un intervalle de confiance pour notre estimation.
Nous avons besoin d'un générateur de nombre aléatoire pour ce TP, et nous allons reprendre Mersenne Twister de Makoto Matsumoto du TP précédent.

Simulation de PI

Pour simuler la valeur de PI à l'aide de la méthode de Monte Carlo, il faut générer deux nombres aléatoires *x1* et *x2* compris entre 0 et 1. Puis compter le nombre de couple de points, tel que $x1^2 + x2^2 < 1$ => (x1, x2) coordonnées d'un point à l'intérieur d'un quart de disque de rayon 1.

En comptant le nombre de points à l'intérieur du disque sur le nombre total de points, on obtient une estimation de l'aire de ce quart de disque. Comme la surface d'un disque : S_disque = PI * R^2 et ici R = 1 => S_disque = PI Or, la simulation donne la valeur de la surface d'un quart de disque : S_quart_disque = S _disque / 4 Donc **PI = 4 * S_quart_disque**

```
/*****************************************************/
/*simPi : calcul la valeur de PI avec methode de    */
/*         Monte Carlo                               */
/*                                                   */
/*Entre : un entier : nombre de point a tirer       */
/*                                                   */
/*Sortie : un entier : valeur de pi calcule         */
/*****************************************************/
double simPi(int nb_point)
{
    //printf("simPi avec n = %d points\n", nb_point);

    double x_rand, y_rand;
    double d = 0;   //distance
    int dedans = 0; //nb de point dans le quart de cercle
    double pi_calcule = 0.;
```

```
    // Tirage de nb_point aleatoire dans l'espace [0..1]²
    for(int i = 0; i < nb_point; ++i)
    {
        x_rand = genrand_real2();
        y_rand = genrand_real2();

        d =  (x_rand * x_rand) + (y_rand * y_rand); // calcul de la distance au
centre
        //d = sqrt(d);  //car x² + y² < R² = 1² = 1
        if (d < 1)  // test du point dans le quart de cercle
        {
            dedans++;
        }
    }
    // disque : S_disque = pi*R² => ici S_quart_disque = S_disque/4 => pi = 4 * S
    pi_calcule = 4 * (double)dedans / (double)nb_point;

    //printf("La valeur de PI calculee est : %f\n", pi_calcule);
    return pi_calcule;
}
```

Resulats :

```
Valeur reelle de PI (avec math.h) : 3.141593

valeur de pi estimee : 3.124000
temps d'execution 0.00008 s pour n = 1000 points


valeur de pi estimee : 3.144720
temps d'execution 0.03476 s pour n = 1000000 points


valeur de pi estimee : 3.141541
temps d'execution 23.65356 s pour n = 1000000000 points
```

On remarque que le temps d'execution est relativement long pour la dernière estimation, cela vient du fait que l'on a compilé sans option d'optimisation.

```
gcc tp3.c -o prog -lm -O2
```

Resultats avec une optimisation O2 :

```
Valeur reelle de PI (avec math.h) : 3.141593

valeur de pi estimee : 3.124000
```

```
temps d'execution 0.00002 s pour n = 1000 points


valeur de pi estimee : 3.144720
temps d'execution 0.01647 s pour n = 1000000 points


valeur de pi estimee : 3.141541
temps d'execution 7.10497 s pour n = 1000000000 points
```

On remarque que le temps d'execution à été divisé par un peu plus de 3.

Calcul moyenne et intervalle de confiance

On souhaite maintenant calculer la moyenne des estimations de PI. On souhaite également donner un intervale de confiance à 95% pour notre estimation. Pour cela on utilise les formules et tables données en annexe.

```
void mean_simPi(int nb_launch, int nb_point)
{
    double pi;
    double mean_pi = 0;
    double valeur[nb_launch];

    for (int i = 0; i < nb_launch; ++i)
    {
        pi = simPi(nb_point);
        valeur[i] = pi;
        mean_pi += pi;
    }
    mean_pi /= nb_launch;

    /*** Calcul intervale de confiance ***/

    //Calcul de S²(n)
    double s_n_carre = 0;
    for (int k = 0; k < nb_launch; k++)
    {
        s_n_carre += (valeur[k] - mean_pi) * (valeur[k] - mean_pi);
    }
    s_n_carre /= nb_launch - 1;

    // Calcul du rayon de confiance
    double tab_t_n_1[3] = {2.228, 2.086, 2.042}; // n = 10, 20, 30
    double t_n_1 = tab_t_n_1[nb_launch / 10 - 1];
    double r = t_n_1 * sqrt(s_n_carre / nb_launch);

    double ecart = fabs(mean_pi - M_PI);

    printf("\nPour %d simulation de pi\n", nb_launch);
```

```
    printf("Avec %d points \n", nb_point);
    printf("\nValeur moyenne de pi : %f\n", mean_pi);
    printf("ecart calcul : %f\n", ecart);
    printf("ecart relatif : %f\n", ecart / nb_launch);

    printf("\nIntervale de confiance a 95%%\n");
    printf("pi : [ %f - %f; %f + %f ]\n", mean_pi, r, mean_pi, r);
    printf("pi : [ %f ; %f ] a 95%% de confiance\n", mean_pi - r, mean_pi + r);
}
```

Résultats :

```
Valeur reelle de PI (avec math.h) : 3.141593

valeur de pi estimee : 3.124000
temps d'execution 0.00008 s pour n = 1000 points


Pour 10 simulation de pi
Avec 1000 points

Valeur moyenne de pi : 3.158000
ecart calcul : 0.016407
ecart relatif : 0.001641

Intervale de confiance a 95%
pi : [ 3.158000 - 0.032738; 3.158000 + 0.032738 ]
pi : [ 3.125262 ; 3.190738 ] a 95% de confiance


Pour 20 simulation de pi
Avec 1000 points

Valeur moyenne de pi : 3.134800
ecart calcul : 0.006793
ecart relatif : 0.000340

Intervale de confiance a 95%
pi : [ 3.134800 - 0.021265; 3.134800 + 0.021265 ]
pi : [ 3.113535 ; 3.156065 ] a 95% de confiance


Pour 30 simulation de pi
Avec 1000 points

Valeur moyenne de pi : 3.142400
ecart calcul : 0.000807
ecart relatif : 0.000027

Intervale de confiance a 95%
pi : [ 3.142400 - 0.019987; 3.142400 + 0.019987 ]
pi : [ 3.122413 ; 3.162387 ] a 95% de confiance
```

Réiterer l'expérience de simulation plusieurs fois n'augmente pas la précision des décimales de PI, mais permet de calculer un intervalle de confiance pour notre valeur moyenne de PI.