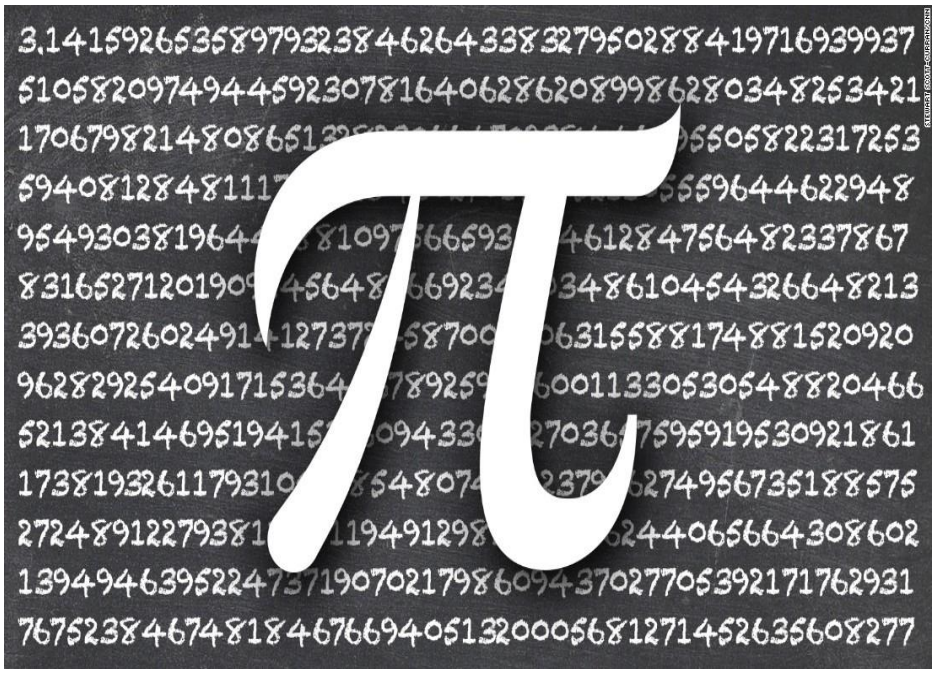


Simulation de Monte Carlo & Intervalles de Confiance

TP3 DE SIMULATION

Anaël Courjaud | S1 ZZ2 F2 | 07/11/2022

Introduction

Comme découvert dans le TP2, je vais désormais toujours privilégier l'utilisation du Mersenne Twister dès qu'il sera question de générer des nombres aléatoires. Surtout pour les applications scientifiques.

Dans ce TP j'ai donc choisi la compilation séparée avec un module et son header entièrement dédiés au Mersenne Twister (mersenneTwister.c et mersenneTwister.h). J'ai inclus general.h (qui contient les bibliothèques et les macros nécessaires à l'ensemble du code) à mersenneTwister.h puis j'inclus mersenneTwister.h à mersenneTwister.c et à main.c.

Ainsi je dispose dans mon main.c des fonctions de génération de nombres aléatoires du Mersenne Twister ainsi que des bibliothèques et des macros importantes.

Dans le main, il y a plusieurs petits blocs de code, séparés par 2 retours à la ligne. Dès que vous voudrez exécuter un certain bout de code, il faudra tout commenter (sauf l'initialisation et le return) et décommenter uniquement le bout de code concerné. C'est important pour des questions d'initialisation du twister et pour obtenir les mêmes résultats que moi.

```c
int main(void)
{
    unsigned long init[4]={0x123, 0x234, 0x345, 0x456}, length=4;
    init_by_array(init, length);

    // 1ere partie de la question 1
    // simPI(30000, true);


    // 2eme partie de la question 1
    // printf("nombre de points nécessaires : %ld\n", simPIresearch(0.001, 100000000));


    // 1ere partie de la question 2
    // int n = 40;
    // int nbrPoints = 1000000;
    // double resultats[n];
    // printf("meanPI pour n = %d et nbrPoints = %d est de %lf\n", n, nbrPoints, calculerMeanPI(n, nbrPoints, resultats));


    // 2eme partie de la question 2
    // int n = 40;
    // int nbrPoints = 1000000;
    // double resultats[n] = {0};
    // double meanPI = calculerMeanPI(n, nbrPoints, resultats);
    // double erreurAbsolue = meanPI - M_PI;
    // if(erreurAbsolue < 0){
    //     erreurAbsolue *= -1;
    // }
    // printf("meanPI pour n = %d et nbrPoints = %d est de %lf\nErreur absolue : %lf\nErreur relative : %lf\n", n, nbrPoints, meanPI, erreurAbsolue, erreurAbsolue/M_PI);


    // question 3
    int n = 40;
    int nbrPoints = 1000;
    double meanPI;
    double confidenceRadius;
    calculerMeanEtConfidenceRadius(n, nbrPoints, &meanPI, &confidenceRadius);
    printf("L'intervalle de confiance à 95 pourcents pour n = %d et nbrPoints = %d est de [%lf, %lf]\n", n, nbrPoints, meanPI - confidenceRadius, meanPI + confidenceRadius);


    return 0;
}
```

Figure 1 : Capture d'écran du main dans main.c

Question 1

1. Réalisation du code de simPI

Pour la question 1 dont le but est d'estimer PI par la méthode de Monte-Carlo, j'ai codé la fonction simPi(int nbrPoints) directement au-dessus du main dans main.c.

simPi(int nbrPoints) va générer nbrPoints points dans le carré [0;1]X[0;1]. On utilise donc 2 fois genrand_real1() par point pour générer les coordonnées xPoint et yPoint de chaque point.

Ensuite, pour chaque point généré, on va vérifier si il appartient au quart-de-disque de rayon 1 et incrémenter de 1 la variable estDedans le cas échéant. Pour cela, il suffit de vérifier la condition $xPoint^2 + yPoint^2 <= 1$ (Théorème de Pythagore).

Pour finir on utilise le fait que le rapport entre nbrPoints et estDedans converge (quand nbrPoints tend vers l'infini) vers le rapport entre la surface de "l'aire de génération" (S = 1) et la surface du quart-de-disque de rayon 1 (Sprime = PI/4).

Or cela revient à 4*estDedans/nbrPoints converge (quand nbrPoints tend vers l'infini) vers PI. On va donc écrire dans la console le résultat de cette opération afin de l'étudier en fonction de nbrPoints.

```c
double simPI(int nbrPoints, bool afficherExplications){

    int nbrDedans = 0;
    double xPoint, yPoint;

    for(int i=0; i<nbrPoints; i++){

        xPoint = genrand_real1();
        yPoint = genrand_real1();

        if(xPoint*xPoint + yPoint*yPoint <= 1.0){ //vérifie si un point de coordonnées x,y appartient au disque de rayon 1 et de centre l'origine
            nbrDedans++;
        }
    }

    double experimentalPI = 4.0 * (double)nbrDedans / (double)nbrPoints;

    if(afficherExplications){
        printf("Sur N = %d points générés aléatoirement dans le carré de 0 à 1, M = %d
        appartiennent au quart-de-disque de centre l'origine et de rayon 1\n", nbrPoints, nbrDedans);
        printf("La surface de 'l'aire d'apparition' des points est de 1 (S = 1). La surface du quart-de-disque est de PI/4 (Sprime = PI/4).\n");
        printf("Or, avec N tendant vers l'infini, M/N converge vers Sprime/S et donc 4*M/N converge vers PI\n");
        printf("Donc PI est égal à environ 4*%d/%d = %f \n", nbrDedans, nbrPoints, experimentalPI);
    }

    return experimentalPI;
}
```

Quelques résultats de simPI, pour nbrPoints = 1000, 1 000 000 et 1 000 000 000 :

```
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
Sur N = 1000 points générés aléatoirement dans le carré de 0 à 1, M = 781 appartiennent au quart-de-disque de centre l'origine et de rayon 1
La surface de 'l'aire d'apparition' des points est de 1 (S = 1). La surface du quart-de-disque est de PI/4 (Sprime = PI/4).
Or, avec N tendant vers l'infini, M/N converge vers Sprime/S et donc 4*M/N converge vers PI
Donc PI est égal à environ 4*781/1000 = 3.124000
```

```
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
Sur N = 1000000 points générés aléatoirement dans le carré de 0 à 1, M = 786190 appartiennent au quart-de-disque de centre l'origine et de rayon 1
La surface de 'l'aire d'apparition' des points est de 1 (S = 1). La surface du quart-de-disque est de PI/4 (Sprime = PI/4).
Or, avec N tendant vers l'infini, M/N converge vers Sprime/S et donc 4*M/N converge vers PI
Donc PI est égal à environ 4*786190/1000000 = 3.144760
```

```
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
Sur N = 1000000000 points générés aléatoirement dans le carré de 0 à 1, M = 785386377 appartiennent au quart-de-disque de centre l'origine et de rayon 1
La surface de 'l'aire d'apparition' des points est de 1 (S = 1). La surface du quart-de-disque est de PI/4 (Sprime = PI/4).
Or, avec N tendant vers l'infini, M/N converge vers Sprime/S et donc 4*M/N converge vers PI
Donc PI est égal à environ 4*785386377/1000000000 = 3.141546
```

2. Analyse des résultats de simPI

Après avoir testé avec nbrPoints égal à 1000, 1 000 000 et 1 000 000 000, on remarque que plus nbrPoints est grand, plus le résultat expérimental est proche de la valeur théorique de PI (et plus les calculs sont longs : environ 10 secondes pour 1 000 000 000). C'est donc cohérent avec le fait que la vitesse de convergence est égale à sqrt(nbrPoints) (dis dans le cours et rappelé dans l'énoncé).

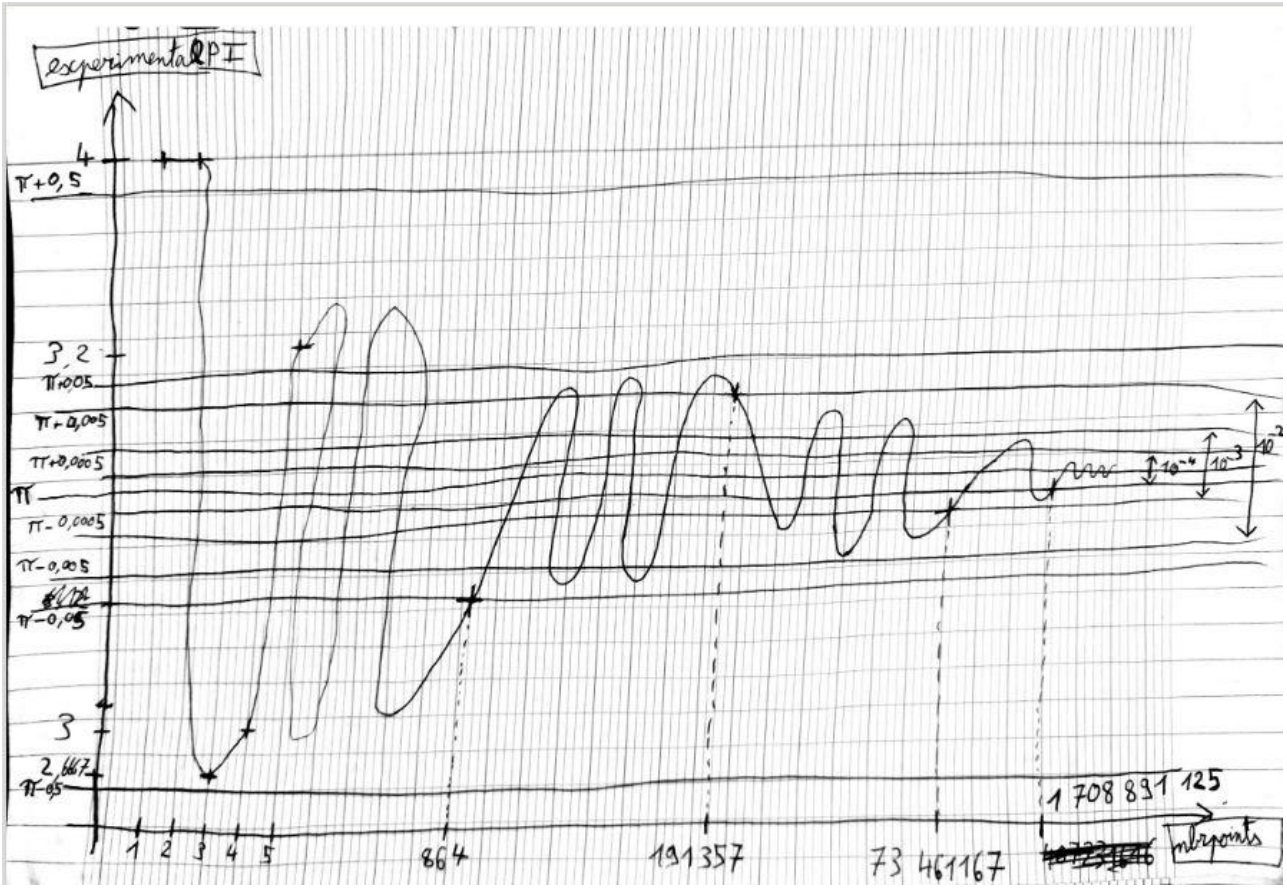
J'ai donc cherché à partir de quelle valeur de nbrPoints, on peut dire que on a trouvé PI à 10^-n près (n un entier naturel).

simPi(nbrPoints) converge vers PI en fonction de nbrPoints, donc si on se rappelle les propriétés des limites, on a que : pour tout n entier naturel, il existe un unique nbrPoints à partir duquel

PI-(10^-n)/2 <= simPI <= PI+(10^-n)/2.

J'ai donc codé une fonction simPIresearch(double précision voulue, long longueurTube) qui peut trouver expérimentalement ce fameux nbrPoints pour chaque précision voulue. Pour confirmer expérimentalement que la fonction a trouvé le bon nbrPoints, elle vérifie par le calcul que les longueurTube prochaines itérations respectent bien la formule de la limite (donc qu'elles appartiennent chacunes bien au "tube" centré sur PI et large de la précision voulue).

J'ai fait ce schéma nul à la main pour présenter grossièrement les résultats des calculs de mon ordinateur et pour donner une idée du comportement de simPI(nbrPoints). Oui je sais c'est très sale et rien n'est à l'échelle mais je trouve que ça représentait assez bien mes observations.



J'ai mis longueurTube en variable pour pouvoir l'adapter en fonction de la précision voulue. Par exemple pour calculer à la précision 10^-1, une longueurTube de 100000 est très largement suffisante alors que pour calculer la précision 10^-4, une longueurTube de 10000000000 a été nécessaire et mon ordinateur a mis dix bonnes minutes à calculer tout ça.

Le choix arbitraire de la valeur de longueurTube n'est pas très rigoureux d'un point de vue mathématiques mais bon, en informatique on a pas accès à l'infini alors voilà.

Par exemple, d'après la fonction simPIresearch(précisionVoulue, longueurTube) (avec une longueurTube adaptée pour chaque précision recherchée), simPI entre définitivement dans l'intervalle [PI-0.05 ; PI+0.05] (précision 10^-1) pour nbrPoints >= 864 (voir schéma suivant).

Pour résumer, avec mon initialisation du Mersenne Twister, j'ai trouvé que :

- Pour trouver expérimentalement la valeur de PI à 10^-2 près, il fallait générer au minimum 191 357 points.
- Pour 10^-3, minimum 73 461 167 points.
- Pour 10^-4, minimum 1 708 891 125 points.
- Et pour 10^-5 et au-delà, je doute que mon ordinateur puisse y arriver en moins d'une journée entière de calculs avec cette méthode (la méthode de Monte-Carlo).

Question 2

Dans un premier temps, j'ai simplement codé la fonction calculerMeanPI(int n, int nbrPoints) qui retourne la moyenne des n experimentalPI calculés avec chacuns des nbrPoints points. J'ai l'impression que le résultat pour tout n et nbrPoints entiers naturels non nuls est le même que simPI(n*nbrPoints) (ce qui est logique vu l'utilisation qui est faite du Mersenne Twister).

```
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
Sur N = 30000 points générés aléatoirement dans le carré de 0 à 1, M = 23548 appartiennent au quart-de-disque de centr
La surface de 'l'aire d'apparition' des points est de 1 (S = 1). La surface du quart-de-disque est de PI/4 (Sprime = P
Or, avec N tendant vers l'infini, M/N converge vers Sprime/S et donc 4*M/N converge vers PI
Donc PI est égal à environ 4*23548/30000 = 3.139733
```

```
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
meanPI pour n = 30 et nbrPoints = 1000 est de 3.139733
```

J'ai ensuite testé la fonction avec n égal à 10, 20 ou 40 et avec nbrPoints égal à 1000, 1 000 000 ou 1 000 000 000.

```
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progMT  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
meanPI pour n = 10 et nbrPoints = 1000 est de 3.148000
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progMT  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
meanPI pour n = 20 et nbrPoints = 1000 est de 3.140200
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progMT  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
meanPI pour n = 40 et nbrPoints = 1000 est de 3.142400
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progMT  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
meanPI pour n = 10 et nbrPoints = 1000000 est de 3.141286
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progMT  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
meanPI pour n = 20 et nbrPoints = 1000000 est de 3.141137
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progMT  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
meanPI pour n = 40 et nbrPoints = 1000000 est de 3.141186
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progMT  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
meanPI pour n = 10 et nbrPoints = 1000000000 est de 3.141563
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progMT  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
meanPI pour n = 20 et nbrPoints = 1000000000 est de 3.141586
```

Et pour finir, j'ai proposé une portion de code dans le main qui permet de calculer et d'afficher l'erreur absolue et relative du résultat de la fonction calculerMeanPI.

```
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progMT
meanPI pour n = 40 et nbrPoints = 1000000 est de 3.141186
Erreur absolue : 0.000406
Erreur relative : 0.000129
```

Question 3

Afin de calculer l'intervalle de confiance à 95% autour de la moyenne des PI simulés par la méthode de Monte-Carlo, j'ai codé la fonction calculerMeanEtConfidenceRadius (on aime beaucoup le franglais ici) qui prend en paramètre n le nombre de réplications voulus et nbrPoints le nombre de points à générer pour chacunes des réplications. La fonction va « renvoyer » (via un système de pointeurs passés en paramètre) la moyenne des n réplications (meanPI) ainsi que le rayon de confiance à 95% (confidenceRadius). Ainsi je peux afficher l'intervalle de confiance à 95% associé au couple (n, nbrPoints).

Pour cela, j'ai suivi les indications données en annexe : à commencer par transcrire le tableau suivant afin de pouvoir l'utiliser en temps voulus dans la fonction.

Table 1: Values of $t_{n-1,1-\alpha/2}$ of a Student law starting with $\alpha = 0.05$ depending on n experiments.

1≤n≤10	$t_{n-1,1-\alpha/2}$	11≤n≤20	$t_{n-1,1-\alpha/2}$	21≤n≤30	$t_{n-1,1-\alpha/2}$	n>30	$t_{n-1,1-\alpha/2}$
1	12.706	11	2.201	21	2.080	40	2.021
2	4.303	12	2.179	22	2.074	80	2.000
3	3.182	13	2.160	23	2.069	120	1.980
4	2.776	14	2.145	24	2.064	+∞	1.960
5	2.571	15	2.131	25	2.060		
6	2.447	16	2.120	26	2.056		
7	2.365	17	2.110	27	2.052		
8	2.308	18	2.101	28	2.048		
9	2.262	19	2.093	29	2.045		
10	2.228	20	2.086	30	2.042		

Après avoir calculé Xbarre (meanPI) et avoir stocké les n Xi (réplicats de la simulation de PI) dans le tableau resultats[n],

$$\overline{X}(n) = \sum_{i=1}^{n} X_i / n$$

j'ai calculé l'estimateur sans biais de la variance de l'échantillon statistique(les n réplicats (Xi)) selon la formule suivante :

$$S^2(n) = \frac{\sum_{i=1}^{n} \left[X_i - \overline{X}(n)\right]^2}{n-1}$$

Une fois ceci fait, il a suffit de calculer le rayon de confiance à 95% en récupérant la valeur associée à n (taille de l'échantillon statistique) dans le tableau tableOfT et en appliquant la valeur de l'estimateur sans biais de la variance calculé précédemment. La formule suivante détaille ce calcul :

$$R = t_{n-1,1-\alpha/2} \times \sqrt{\frac{S^2(n)}{n}}$$

Enfin, dans le main cette fois puisque la fonction s'est occupée de « retourner » Xbarre et R (càd meanPI et confidenceRadius). Il ne manquait plus qu'à afficher le tout dans la console sous la forme suivante :

$$\left[\bar{X} - R, \bar{X} + R\right]$$

Et au final, ça donne les résultats suivants :

```
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progTP3  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progTP3
L'intervalle de confiance à 95 pourcents pour n = 40 et nbrPoints = 1000000 est de [3.140640, 3.141733]
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progTP3  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progTP3
L'intervalle de confiance à 95 pourcents pour n = 20 et nbrPoints = 1000 est de [3.118871, 3.161529]
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progTP3  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progTP3
L'intervalle de confiance à 95 pourcents pour n = 40 et nbrPoints = 100000000 est de [3.141514, 3.141617]
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ make
gcc main.c mersenneTwister.c mersenneTwister.h general.h -o progTP3  -Wall -Wextra -MMD -Og -lm -g
ancourjaud@ada:~/TPsimuS3/TP3simu.05.11.2022$ ./progTP3
L'intervalle de confiance à 95 pourcents pour n = 40 et nbrPoints = 1000 est de [3.127070, 3.157730]
```

Il est préférable de mettre un n compris entre 30 et 40 car c'est là que l'intervalle de confiance commence à devenir vraiment solide statistiquement et comme d'habitude, plus nbrPoints est grand, plus l'intervalle de confiance va être étroit, ce qui permet d'estimer plus précisément PI.

Conclusion

Dans ce TP, on a pu constater la robustesse de la méthode de Monte-Carlo via une application sur le calcul des décimales de PI. On a aussi pu constater de son temps de calcul qui peut rapidement croître de façon exponentiel en fonction de la précision voulue. Ce problème provient de la vitesse de convergence qui est de l'ordre de la racine carrée du total d'éléments générés. En revanche la méthode de Monte-Carlo permet de produire des résultats dans des espaces avec beaucoup de dimensions ce qui est pratique par rapport à d'autres techniques plus rapides, mais qui ne fonctionnent pas dans de tels espaces.

On a aussi pu voir l'avantage procuré par la réalisation de plusieurs expériences indépendantes ce qui va permettre d'appliquer de façon fiable les lois de la statistique (à partir de 30 réplications) et d'ainsi pouvoir déterminer des intervalles de confiance (en l'occurrence à 95%) qui vont aider à trouver des estimations fiables de la valeur de PI avec moins de temps de calcul.