ESTIMATION PSEUDO ALÉATOIRE DE PI

Introduction

Le but de ce projet est de déterminer la valeur de Pi en utilisant des nombres pseudo-aléatoires. Pour atteindre cet objectif, nous allons utiliser l'algorithme de Mersenne Twister, qui est une méthode courante de génération de nombres pseudo-aléatoires.

La méthode employée consiste à générer des nombres aléatoires X et Y compris entre 0 et 1. Pour ensuite appliquer des règles trigonométriques afin de calculer la surface d'un quart de cercle. En multipliant cette surface par 4, on obtient la surface totale du cercle, ce qui permet de déterminer la valeur de Pi.

Cette approche est une méthode connue pour estimer la valeur de Pi. Elle est basée sur le fait que la surface d'un cercle est directement proportionnelle à son rayon au carré. En utilisant des nombres pseudo-aléatoires pour estimer la surface d'un quart de cercle, cette méthode fournit une approximation de la valeur de Pi.

L'algorithme de Mersenne Twister est utilisé en raison de sa rapidité et de sa fiabilité dans la génération de nombres pseudo-aléatoires. Cela permet à l'équipe de générer rapidement un grand nombre de nombres aléatoires pour améliorer la précision de leur estimation de Pi.

```c
#include <stdio.h>
#include <math.h>

/* ------------------------------------------------ */
/* estimPi renvoie une estimation de pi*/
/*
/* En entrée: inNbPts un nombre de points de type long */
/*
/* En sortie: une estimation de pi */
/* ------------------------------------------------ */

double estimPi(long inNbPts)
{
    int cpt = 0;
    double Xr;
    double Yr;

    for(int i = 0; i<inNbPts;i++)
    {
        Xr = genrand_real1();
        Yr = genrand_real1();
        if((Xr*Xr) + (Yr*Yr) < 1 ){cpt++;}

    }
    return((double) cpt/(double)inNbPts *4);
}


/* ------------------------------------------------ */
/* meanPi renvoie une moyenne des estimations de pi */
/*
/* En entrée: nombre d'iterations de type int*/
/*
/* En sortie: une moyenne des estimations de pi*/
/* ------------------------------------------------ */
double meanPi(int iter)
```

```
37 {
38
39     double Bin[iter];
40     double somme = 0.0;
41     for(int i = 0; i < iter; i++)
42     {
43         Bin[i] = estimPi(1000000);
44         somme +=Bin[i];
45
46         //printf(" Pi = %f  ",Bin[i]); // debugage
47         //printf("total = %f  %d \n",somme , i
   );   // debugage
48     }
49     return (somme/iter);
50
51
52 }
53
54 /* ------------------------------------------------
   ------------------- */
55 /* relativeError renvoie l'erreure relative entre
   le pi estimé et Pi*/
56 /*
57 /* En entrée: rien*/
58 /*
59 /* En sortie: l'erreure relative entre pi estimé et
    Pi*/
60 /* ------------------------------------------------
   ------------------- */
61
62 double relativeError()
63 {
64     return((meanPi(30) - M_PI)/M_PI);
65 }
66
67 /* ------------------------------------------------
   ------------------- */
68 /* StudentLaw renvoie l'intervale de confiance*/
69 /*
70 /* En entrée: n de type int */
71 /*
72 /* En sortie: l'intervale de confiance */
```

```
 73 /* ---------------------------------------------
    ---------------------- */
 74
 75 double StudentLaw(int n)
 76 {
 77     double valuesTab[30]= {12.706,4.303,3.182,2.
    776,2.571,2.447,2.365,2.308,   // Tabeau de la
    lois de student
 78                            2.262,2.228,2.201,2.179
    ,2.160,2.145,2.131,2.120,
 79                            2.110,2.101,2.093,2.086
    ,2.080,2.074,2.069,2.064,
 80                            2.060,2.056,2.052,2.048
    ,2.045,2.042};
 81
 82     double value = valuesTab[n-1];
 83     double tabPi[n];
 84     double mean = meanPi(n);
 85     double somme = 0.0;
 86     double S2 = 0.0;
 87     double R = 0.0;
 88
 89     if (n>30 && n<=40) {value = 2.021;}
 90     if (n>40 && n<=80) {value = 2.000;}
 91     if (n>80 && n<=120) {value = 1.980;}
 92     if (n>120) {value = 1.960;}
 93
 94     for (int i = 0; i <n; i++)
 95     {
 96         tabPi[i] = estimPi(1000000000);
 97
 98         somme += (tabPi[i] - mean)* (tabPi[i] -
    mean);
 99
100     }
101
102
103     S2 = (somme/(n-1));
104
105
106     R = (value * (sqrt(S2/n)));
107
```

```
108     return(R);
109 }
110 
111 /* ---------------------------------------------
    ---------------------- */
112 /* intervaleConfiance affiche l'intervale de
    confiance entre deux estimations de pi*/
113 /*
114 /* En entrée: n de type int */
115 /*
116 /* En sortie: void*/
117 /* ---------------------------------------------
    ---------------------- */
118 void intervaleConfiance(int n)
119 {
120     double pi = meanPi(30);
121     double student = StudentLaw(n);
122     printf("Notre intervale de confiance est entre
     : %.10lf et %.10lf avec  n = %d \n", pi + student
    , pi - student, n);
123 }
124 
125 /* ---------------------------------------------
    ---------------------- */
126 /* graphe permet de faire un graphe des intervales
     de confiance*/
127 /*
128 /* En entrée: x et y entre lesquelles on va
    calculer l'intervale de confiance*/
129 /*
130 /* En sortie: void*/
131 /* ---------------------------------------------
    ---------------------- */
132 void graphe(int x,int y)
133 {
134     for(int i = x; i <y; i++)
135     {
136         intervaleConfiance(i);
137     }
138 }
```