Compte rendu du TP2 de simulation

Mathieu VILLEDIEU DE TORCY

Sommaire :

Introduciton

L'objectif du TP est de générer des nombres aléatoires à l'aide du générateur Mersenne Twister. Par la suite on utilisera ce générateur pour simuler différentes variables aléatoires.

Après avoir récupérer l'implementation en C, on test le résultat obtenue pour garantir la répétabilité du générateur.

```
$./prog > rest.txt
$ diff  rest.txt mt19937ar.out ; echo $?
0
```

Le programme génère bien le même fichier que celui fournit comme réference.

Génération de nombres aléatoires uniformes entre A et B

L'objectif de cette partie est de générer un nombre aléatoire en deux réels A et B, tel que A < B. Pour cela on utilise le générateur Mersenne Twister des nombres aléatoires. Celui-ci fournit un nombre aléatoire *x* entre 0 et 1 auxquels on applique la fonction $f : [0..1] \rightarrow [A..B], x \rightarrow A + (B - A) * x$ pour le translater dans l'intervalle souhaité.

```
/* ------------------------------------------------------------ */
/* uniform : retourne un nombre aléatoire entre deux réels a et b */
/*                                                              */
/* Entrée : deux réels a et b                                   */
/* Sortie : un nombre aléatoire entre a et b                    */
/* ------------------------------------------------------------ */
double uniform(double a, double b)
```

```
{
    double rand = genrand_real2();
    return a + (b - a) * rand;
}
```

Fonction de test pour la génération de nombres aléatoires uniformes entre deux réels.

```
/* ------------------------------------------------------------- */
/* test_uniform : test differentes valeurs pour n de tirage      */
/*                de la fct uniform(a, b)                        */
/* Entrée : deux réels a et b                                    */
/* Sortie : un nombre aléatoire entre a et b                     */
/* ------------------------------------------------------------- */
void test_uniform(float a, float b)
{
    int n = 1000000;      //Nombre de tirage
    float temp[n];        //Tableau stockant les differentes valeurs
    float moyenne = 0;

    float val = uniform(a, b);
    temp[0] = val;

    float max = val;
    float min = val;

    for (int i = 1; i < n; i++)
    {
        val = uniform(a, b);
        temp[i] = val;
        moyenne += val;
        if (val > max)
            max = val;
        if (val < min)
            val = min;
    }
    moyenne /= n;

    printf("\nPour %d tirage sur une loi uniforme entre %.2f et %.2f\n", n, a, b);
    printf("moyenne = %.2f, vrai moyenne (%.2f + %.2f)/2 = %.2f\n", moyenne, a, b,
(a + b) / 2);
    printf("max = %.2f\n", max);
    printf("min = %.2f\n", min);
    printf("\n");
}
```

Dans le `main` on execute :

```
    double a = -89.2;
    double b = 56.7;
```

```
    printf("Generate pseudo-random numbers between %3.2f and %3.2f : %3.2f \n", a,
b, uniform(a, b));
    test_uniform(a, b);
```

Résultat :

```
$ gcc mt19937ar.c -o prog -lm
$ ./prog
Generate pseudo-random numbers between -89.20 and 56.70 : 29.67

Pour 1000000 tirage sur une loi uniforme entre -89.20 et 56.70
moyenne = -16.25, vrai moyenne (-89.20 + 56.70)/2 = -16.25
max = 56.70
```

Reproduction de distributions empiriques discrètes

Il s'agit ici de reproduire une distribution une distribution discrete de 3 classes tel que :

classe:	**A**	**B**	**C**
proba:	0.5	0.1	0.4

Pour simuler la population ayant la distribution indiquée nous allons utiliser les

Pour simuler une population d'individus ayant la distribution indiquée, nous réutilisons la procédure de génération d'un nombre aléatoire uniforme entre 0 et 1 précédente.

```
/* ------------------------------------------------------------ */
/* discrete_distribution : cree un distribution discrete 3classes */
/*                         avec P(A)=0.5; P(B)=0.1; P(C)=0.4      */
/* Entrée : entier n -> nb drawings                             */
/* Sortie : rien                                                */
/* ------------------------------------------------------------ */
void discrete_distribution(int n)
{
    double rand;
    int i;

    int population[3];
    float proba[3];

    //Initialise les differentes population a 0
    for (i = 0; i < 3; i++)
    {
        population[i] = 0;
    }
```

```
    for (i = 0; i < n; ++i)
    {
        rand = genrand_real2();

        if (rand < 0.5)         //Appartient a la classe A
        {
            population[0]++;
        }
        else
        {
            if (rand < 0.6)     // Appartient a la classe B
            {
                population[1]++;
            }
            else                //Appartient a la classe C
            {
                population[2]++;
            }
        }
    }

    printf("\nn = %d drawings\n", n);
    for (i = 0; i < 3; i++)
    {
        proba[i] = (float)population[i] / (float)n;
        printf("population %d = %d et frequence = %f\n", i, population[i],
proba[i]);
    }
}
```

On obtient le résultat suivant pour 1000 et 1 000 000 drawings :

```
n = 1000 drawings
population 0 = 517 et frequence = 0.517000
population 1 = 92 et frequence = 0.092000
population 2 = 391 et frequence = 0.391000

n = 1000000 drawings
population 0 = 500079 et frequence = 0.500079
population 1 = 100293 et frequence = 0.100293
population 2 = 399628 et frequence = 0.399628
```

On remarque bien que plus la valeur de n est grande, ie plus il y de tirages, plus les fréquences calculées se rapprochent des fréquences théorique.

Reproduction de distributions continues

Il est possible de reproduire des distributions continues en inversant leur fonction de distribution. Pour une loi exponentielle négative : R = 1 - e^(x / M) <=> x = - M * ln(1 - R).

R = 1 - e^(4).

avec R le nombre aléatoire entre 0 et 1.

```
/* ------------------------------------------------------------ */
/* negExp :  f^(-1) d'une loi exponentielle négative uniforme   */
/*                                                              */
/* Entrée : un real : moyenne                                   */
/* Sortie : un nb généré aléatoirement suivant                  */
/*          une loi exp negative                                */
/* ------------------------------------------------------------ */
double negExp(double mean)
{
    double rand = genrand_real2();
    double tmp = 1. - rand;
    return (-mean * log(tmp));
}
```

Pour facilité l'usage et rendre plus visuel les données j'ai créée un fonction traçant grossièrement un histogramme.

```
/* ------------------------------------------------------------------- */
/* trace_histo :  affiche un histogramme d'un tableau de valeurs       */
/*                                                                     */
/* Entrée :   un tableau d'entier                                      */
/*            un entier : taille du tableau                            */
/*            un entier : precision de l'unite de l'histo              */
/* Sortie : rien                                                       */
/* ------------------------------------------------------------------- */
void trace_histo(int bin[], int nb_bin, int precision)
{
    printf("histogramme\n");
    printf("unite/valeurs quantite*precision");
    int quantite;
    int tot = 0;

    for (int i = 0; i < nb_bin; i++)
    {
        //Affiche l'axe des unites/valeurs
        printf("\n[%2d..%2d] ", i, i + 1);                  //exponentiel
negative
        //printf("\n%3d ", i + 20);                        //somme lance de 20
des
        //printf("\n[%+.1f..%+.1f] ", (float)i / 10. - 1.0,
        //                           (float)i / 10. - 0.9); //Box Muller

        quantite = bin[i] / precision;

        printf("%3d*%d ", quantite, precision);
```

```
        for (int j = 0; j < quantite; j++)
        {
            printf("■");
        }
    }
    printf("\n");
}
```

Je test maintenant la fonction `negExp` avec :

```
/* ------------------------------------------------------------ */
/* testnegExp :  test negExp en affichant un histo              */
/*                                                              */
/* Entrée : un real : mean                                      */
/*          un entier n : nombre de tirage                      */
/*          un entier precision_histo                           */
/* Sortie : rien                                                */
/* ------------------------------------------------------------ */
void test_negExp(double mean, int n, int precision_histo)
{
    double average = 0.;
    double tirage;
    int bin[20];        //tableau comptant les valeurs entre un certain intervalle

    //Initialisation du tableau
    for (int k = 0; k < 20; k++)
        bin[k] = 0;

    for (int i = 0; i < n; ++i)
    {
        tirage = negExp(mean);
        average += tirage;
        if (tirage > 20.)
        {
            bin[19]++;
        }
        else
            bin[(int)tirage]++;
    }
    average /= n;

    printf("\n\nnegExp(%.3f) -> moyenne calculee est : %.3f pour %d tirages\n",
mean, average, n);
    trace_histo(bin, 20, precision_histo);
}
```

Et pour `test_negExp(10, 1000, 10);` et `test_negExp(10, 1000000, 1000);` cela donne :

```
egExp(10.000) -> moyenne calculee est : 9.878 pour 1000 tirages
histogramme
unite/valeurs quantite*precision
[ 0.. 1]   8*10
[ 1.. 2]   8*10
[ 2.. 3]   8*10
[ 3.. 4]   7*10
[ 4.. 5]   5*10
[ 5.. 6]   6*10
[ 6.. 7]   6*10
[ 7.. 8]   5*10
[ 8.. 9]   2*10
[ 9..10]   3*10
[10..11]   4*10
[11..12]   3*10
[12..13]   3*10
[13..14]   1*10
[14..15]   2*10
[15..16]   2*10
[16..17]   2*10
[17..18]   2*10
[18..19]   0*10
[19..20]  14*10


negExp(10.000) -> moyenne calculee est : 9.996 pour 1000000 tirages
histogramme
unite/valeurs quantite*precision
[ 0.. 1]  94*1000
[ 1.. 2]  85*1000
[ 2.. 3]  78*1000
[ 3.. 4]  70*1000
[ 4.. 5]  64*1000
[ 5.. 6]  57*1000
[ 6.. 7]  52*1000
[ 7.. 8]  47*1000
[ 8.. 9]  42*1000
[ 9..10]  38*1000
[10..11]  34*1000
[11..12]  31*1000
[12..13]  28*1000
[13..14]  25*1000
[14..15]  23*1000
[15..16]  21*1000
[16..17]  19*1000
[17..18]  17*1000
[18..19]  15*1000
```

```
[19..20] 149*1000
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

La représentation de la distribution sur le diagramme est plus fidèle à la réalité dans le deuxième cas avec un nombre de tirages élevés, tout comme la moyenne calculée qui tend vers 10. Les défauts de l'histogramme sont :

- les ■ traçant les barres représentent une quantité fixe, ainsi dans le premier histogramme la ligne [18..19] 0*10 n'affiche rien, mais cela ne veut pas dire que le nombre de valeurs dans cette intervalle est nulle mais qu'il est < 10.
- la dernière ligne [19..20] représente l'intervalle [19..+∞[. Cela explique la grande taille de la barre.

Simulation de loi de distribution non inversible

Somme de lancés de dés

Voici la fonction permettant de simuler le lancé d'un à 6 faces non truqué :

```
/* -------------------------------------------------------------- */
/* dice_6 :  genere un nb aleatoire uniforme entre 1 et 6          */
/*                                                                  */
/* Entrée : rien                                                    */
/*                                                                  */
/* Sortie : un entier : nb aleatoire uniforme entre 1 et 6          */
/* -------------------------------------------------------------- */
int dice_6()
{
    double rand = uniform(1, 7);
    return (int)rand;
}
```

La fonction permettant de sommer le résultat de *m* lancés de dé :

```
/* -------------------------------------------------------------- */
/* dice_sum :  somme du resultat de m lance de des                 */
/*                                                                  */
/* Entrée : un entier : nombre de lance de de                       */
/*                                                                  */
/* Sortie : un entier : somme du resultat de m lance de des         */
/* -------------------------------------------------------------- */
int dice_sum(int m)
{
    int dice = 0;

    for (int i = 0; i < m; ++i)
    {
        dice += dice_6();
```

```
    }
    return dice;
}
```

On simule l'expérience plusieurs fois pour calculer une moyenne et compter l'occurrence des valeurs obtenues pour voir la courbe en cloche.

```
/* ------------------------------------------------------------ */
/* repeat_sum_dice :  repete plusieur fois la somme du res de des */
/*                    trace l'histogramme des occurrences       */
/*                    et calcul la moyenne                      */
/*                                                              */
/* Entrée : un entier n : nombre d'iteration de l'experience    */
/*          un entier histo_precision : precision pour histo    */
/*          un entier m : nombre de lance de des pour la somme  */
/*                                                              */
/* Sortie : rien                                                */
/* ------------------------------------------------------------ */
void repeat_sum_dice(int n, int histo_precision int m)
{
    int score_max = m * 6;     // Valeur maximale de score possible
    int score[score_max + 1];  // Tableau comptant les occurences des scores
    int moyenne = 0;           // Moyenne

    // Initialisation du tableau des occurences
    for (int k = 0; k < score_max + 1; k++)
    {
        score[k] = 0;
    }

    int res_sum = 0;

    // Effecture n fois la somme du resultat de m lance de des
    for (int i = 0; i < n; ++i)
    {
        res_sum = dice_sum(m);
        score[res_sum]++;
        moyenne += res_sum;
    }

    moyenne /= n;

    printf("\nmoyenne calculee = %d\n", moyenne);
    trace_histo(score, score_max, histo_precision);
}
```

Voici le résultat obtenue :

```
Somme de m = 20 des et n = 10000 iteretions
moyenne calculee = 69
histogramme
unite/valeurs quantite*precision
  0   0*10
  1   0*10
  2   0*10
  3   0*10
  4   0*10
  5   0*10
  6   0*10
  7   0*10
  8   0*10
  9   0*10
 10   0*10
 11   0*10
 12   0*10
 13   0*10
 14   0*10
 15   0*10
 16   0*10
 17   0*10
 18   0*10
 19   0*10
 20   0*10
 21   0*10
 22   0*10
 23   0*10
 24   0*10
 25   0*10
 26   0*10
 27   0*10
 28   0*10
 29   0*10
 30   0*10
 31   0*10
 32   0*10
 33   0*10
 34   0*10
 35   0*10
 36   0*10
 37   0*10
 38   0*10
 39   0*10
 40   0*10
 41   0*10
 42   0*10
 43   0*10
 44   0*10
 45   0*10
 46   0*10
 47   0*10
 48   0*10
```

```
 49   1*10 ■
 50   1*10 ■
 51   2*10 ■■
 52   2*10 ■■
 53   4*10 ■■■■
 54   4*10 ■■■■
 55   6*10 ■■■■■■
 56  10*10 ■■■■■■■■■
 57  13*10 ■■■■■■■■■■■■
 58  16*10 ■■■■■■■■■■■■■■■
 59  19*10 ■■■■■■■■■■■■■■■■■
 60  21*10 ■■■■■■■■■■■■■■■■■■■
 61  25*10 ■■■■■■■■■■■■■■■■■■■■■■
 62  31*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■
 63  35*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 64  38*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 65  41*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 66  44*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 67  51*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 68  55*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 69  48*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 70  53*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 71  54*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 72  47*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 73  49*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 74  42*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 75  41*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 76  40*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 77  34*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 78  30*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 79  26*10 ■■■■■■■■■■■■■■■■■■■■■■■■■■
 80  22*10 ■■■■■■■■■■■■■■■■■■■■■■
 81  17*10 ■■■■■■■■■■■■■■■■■
 82  14*10 ■■■■■■■■■■■■■■
 83  12*10 ■■■■■■■■■■■■
 84   8*10 ■■■■■■■■
 85   6*10 ■■■■■■
 86   6*10 ■■■■■■
 87   4*10 ■■■■
 88   2*10 ■■
 89   2*10 ■■
 90   1*10 ■
 91   0*10
 92   0*10
 93   0*10
 94   0*10
 95   0*10
 96   0*10
 97   0*10
 98   0*10
 99   0*10
100   0*10
101   0*10
102   0*10
```

```
103   0*10
104   0*10
105   0*10
106   0*10
107   0*10
108   0*10
109   0*10
110   0*10
111   0*10
112   0*10
113   0*10
114   0*10
115   0*10
116   0*10
117   0*10
118   0*10
119   0*10
120   0*10
```

Et le résultat pour 1 000 000 itérations ou l'histogramme se rapproche de la courbe réelle :

```
Somme de m = 20 des et n = 1000000 iteretions
moyenne calculee = 69
histogramme
unite/valeurs quantite*precision
  0   0*1000
  1   0*1000
  2   0*1000
  3   0*1000
  4   0*1000
  5   0*1000
  6   0*1000
  7   0*1000
  8   0*1000
  9   0*1000
 10   0*1000
 11   0*1000
 12   0*1000
 13   0*1000
 14   0*1000
 15   0*1000
 16   0*1000
 17   0*1000
 18   0*1000
 19   0*1000
 20   0*1000
 21   0*1000
 22   0*1000
 23   0*1000
 24   0*1000
 25   0*1000
```

```
 26   0*1000
 27   0*1000
 28   0*1000
 29   0*1000
 30   0*1000
 31   0*1000
 32   0*1000
 33   0*1000
 34   0*1000
 35   0*1000
 36   0*1000
 37   0*1000
 38   0*1000
 39   0*1000
 40   0*1000
 41   0*1000
 42   0*1000
 43   0*1000
 44   0*1000
 45   0*1000
 46   0*1000
 47   0*1000
 48   0*1000
 49   1*1000 ■
 50   1*1000 ■
 51   2*1000 ■■
 52   3*1000 ■■■
 53   4*1000 ■■■■
 54   5*1000 ■■■■■
 55   7*1000 ■■■■■■■
 56   9*1000 ■■■■■■■■■
 57  12*1000 ■■■■■■■■■■■■
 58  15*1000 ■■■■■■■■■■■■■■■
 59  18*1000 ■■■■■■■■■■■■■■■■■■
 60  22*1000 ■■■■■■■■■■■■■■■■■■■■■■
 61  25*1000 ■■■■■■■■■■■■■■■■■■■■■■■■
 62  30*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 63  34*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 64  38*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 65  41*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 66  45*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 67  47*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 68  50*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 69  51*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 70  52*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 71  51*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 72  50*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 73  48*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 74  45*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 75  42*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 76  38*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 77  34*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 78  30*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 79  26*1000 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

```
 80  22*1000 ■■■■■■■■■■■■■■■■■■■■■■
 81  18*1000 ■■■■■■■■■■■■■■■■■■
 82  15*1000 ■■■■■■■■■■■■■■■
 83  12*1000 ■■■■■■■■■■■■
 84   9*1000 ■■■■■■■■■
 85   7*1000 ■■■■■■■
 86   5*1000 ■■■■■
 87   4*1000 ■■■■
 88   3*1000 ■■■
 89   2*1000 ■■
 90   1*1000 ■
 91   1*1000 ■
 92   0*1000
 93   0*1000
 94   0*1000
 95   0*1000
 96   0*1000
 97   0*1000
 98   0*1000
 99   0*1000
100   0*1000
101   0*1000
102   0*1000
103   0*1000
104   0*1000
105   0*1000
106   0*1000
107   0*1000
108   0*1000
109   0*1000
110   0*1000
111   0*1000
112   0*1000
113   0*1000
114   0*1000
115   0*1000
116   0*1000
117   0*1000
118   0*1000
119   0*1000
120   0*1000
```

Box and Muller

La méthode de Box and Muller générer 2 nombres aléatoires suivant une loi Gaussienne centrée réduite en 0, N(0,1).

Elle utilise la formule suivante : x1 = cos(2 * \pi * Rn2)* (-2*ln(Rn1))^0.5 x2 = sin(2 * \pi * Rn2)* (-2*ln(Rn1))^0.5

x1, x2 sont les deux nombres suivant la loi N(0,1) et *Rn1, Rn2* les deux nombres aléatoires.

```c
/* --------------------------------------------------------------- */
/* box_muller :  methode de box et muller                          */
/*                                                                 */
/* Entrée : un entier n : nombre d'iteration de l'experience      */
/*          un entier histo_precision : precision pour histo      */
/*                                                                 */
/* Sortie : 2 entier x1 et x2 suivant N(0,1)                      */
/* --------------------------------------------------------------- */
void box_muller(int n, int histo_precision)
{
    double rn1, rn2;
    double x1, x2;

    //Initialisation tableau de compteur
    int test20bin[20];
    for (int i = 0; i < 20; ++i)
    {
        test20bin[i] = 0;
    }

    for (int k = 0; k < n; k++)
    {
        //Calcul nombre aleatoire suivant une gaussienne N(0,1)
        rn1 = genrand_real2();
        rn2 = genrand_real2();
        x1 = cos(2 * M_PI * rn2) * sqrt(-2 * log(rn1));
        x2 = sin(2 * M_PI * rn2) * sqrt(-2 * log(rn1));

        //Comptabilise les valeurs entre -1 et 1
        if (x1 > -1 && x1 < 1)
            test20bin[(int)((x1 + 1) * 10)]++;
        if (x2 > -1 && x2 < 1)
            test20bin[(int)((x2 + 1) * 10)]++;
    }
    printf("Box and Muller : N(0,1) \n");
    printf("Avec %d tirages et une precision de %d -> ", n, histo_precision);
    trace_histo(test20bin, 20, histo_precision);
}
```

Et résultat, on obtient une courbe :

```
Box and Muller : N(0,1)
Avec 1000000 tirages et une precision de 1000 -> histogramme
unite/valeurs quantite*precision
[-1.0..-0.9]  51*1000 ███████████████████████████████████████████████████
[-0.9..-0.8]  55*1000 ███████████████████████████████████████████████████████
[-0.8..-0.7]  60*1000 ████████████████████████████████████████████████████████████
[-0.7..-0.6]  64*1000 ████████████████████████████████████████████████████████████████
[-0.6..-0.5]  68*1000 ████████████████████████████████████████████████████████████████████
```

```
[-0.5..-0.4]  72*1000
[-0.4..-0.3]  75*1000
[-0.3..-0.2]  76*1000
[-0.2..-0.1]  79*1000
[-0.1..+0.0]  79*1000
[+0.0..+0.1]  79*1000
[+0.1..+0.2]  78*1000
[+0.2..+0.3]  77*1000
[+0.3..+0.4]  74*1000
[+0.4..+0.5]  72*1000
[+0.5..+0.6]  68*1000
[+0.6..+0.7]  64*1000
[+0.7..+0.8]  60*1000
[+0.8..+0.9]  56*1000
[+0.9..+1.0]  51*1000
```