

Marion Carrier
Aurélie Dhenry

SIMULATION TP2

GENERATION DE NOMBRES PSEUDO-ALEATOIRES SUIVANT DIFFERENTES DISTRIBUTIONS

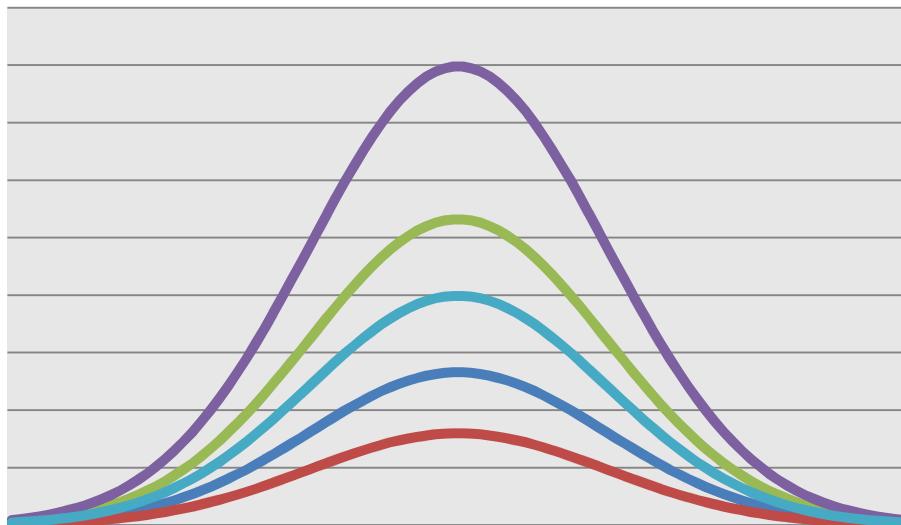


TABLE DES MATIERES

Table des figures.....	3
Introduction	4
1. Exercice préliminaire	4
2. Simulation d'une distribution suivant un histogramme donné	
2.1. Simulation suivant un exemple précis.....	4
2.2. Simulation suivant un histogramme quelconque	6
3. Simulation d'une loi continue inversible : la loi exponentielle	8
4. Simulation d'une loi continue non inversible : la loi normale	
4.1. Méthode de Box et Müller.....	10
4.2. Autres algorithmes	11
4.3. Bibliothèques de génération de nombres pseudo-aléatoires	12
Bibliographie	13
Annexes	

TABLE DES FIGURES

Figure 1 : Histogramme de répartition des espèces selon leurs probabilités	4
Figure 2 : Table de répartition des espèces.....	5
Figure 3 : Répartition moyenne sur 10 simulations à 10 000 tirages, suivant l'histogramme donné en exemple.....	6
Figure 4 : Répartitions théorique et pratique moyenne selon un histogramme à 6 classes, sur 1000 itérations, sur 10 répétitions de l'expérience.....	7
Figure 5 : Répartitions théorique et pratique moyenne selon un histogramme à 4 classes, sur 10 000 itérations, sur 10 répétitions de l'expérience	8
Figure 6 : Simulation de la fonction quantile d'une loi exponentielle de moyenne 10, sur 1000 et 10 000 itérations.....	8
Figure 7 : Représentation graphique d'une simulation d'une répartition suivant une loi exponentielle négative, de moyenne 10, sur 1000 itérations	9
Figure 8 : Représentation graphique d'une simulation moyenne sur 10 expériences d'une loi exponentielle négative, de moyenne 10, sur 10 000 itérations à chaque expérience	9
Figure 9 : Simulation de la fonction quantile d'une loi normale de moyenne 10 et d'écart-type 3, sur 1000 tirages, suivant la méthode de Box et Müller	10
Figure 10 : Représentation graphique d'une simulation d'une distribution suivant une loi normale de moyenne 10 et d'écart-type 3.....	10
Figure 11 : Représentation graphique d'une simulation moyenne sur 10 expériences d'une loi normale de moyenne 10 et d'écart-type 3, sur 10 000 itération à chaque expérience.....	11
Figure 12 : Simulation de la fonction quantile d'une loi normale de moyenne 10 et d'écart-type 3, sur 1000 et 10 000 itérations, suivant la première méthode de réjection adaptée à la loi normale.....	11
Figure 13 : Représentation graphique de deux simulations moyennes sur 10 expériences, d'une loi normale de moyenne 10 et d'écart-type 3, respectivement suivant 1000 et 10 000 itérations par expérience	12

Introduction

Ce TP a pour but de nous faire manipuler des générateurs de nombres aléatoires (nous avons choisi la fonction `rand`), mais non plus uniformément distribués comme lors du premier TP. En effet, nous allons simuler des lois de distributions non uniformes, que ce soit suivant un histogramme donné, ou qu'il s'agisse de lois connues comme la loi exponentielle négative, ou encore une loi normale.

1. Exercice préliminaire

Avant de simuler une distribution non uniforme, faisons un retour au TP1 et testons un code générant des nombres uniformément répartis entre deux nombres A et B, renseignés par l'utilisateur. Nous avons choisi de reprendre pour cela le principe de l'exercice 7 du TP1, c'est-à-dire de tester l'équiprobabilité à l'aide d'autant d'intervalles qu'il y a d'entiers entre A et B.

**** Simulation uniforme ****

```
Borne inferieure : 5  
Borne superieure : 11  
Nombre d'iterations : 1000  
Entre 5 et 6 : 156  
Entre 6 et 7 : 162  
Entre 7 et 8 : 180  
Entre 8 et 9 : 168  
Entre 9 et 10 : 157  
Entre 10 et 11 : 177
```

Que voulez vous faire ?
1 : Recommencer.
0 : Quitter.

**** Simulation uniforme ****

```
Borne inferieure : 78  
Borne superieure : 84  
Nombre d'iterations : 10000  
Entre 78 et 79 : 1657  
Entre 79 et 80 : 1679  
Entre 80 et 81 : 1656  
Entre 81 et 82 : 1694  
Entre 82 et 83 : 1652  
Entre 83 et 84 : 1662
```

Que voulez vous faire ?
1 : Recommencer.
0 : Quitter.

On peut voir que même si la fonction `rand` est un générateur de base, il génère des nombres répartis de manière relativement bien uniforme (cette uniformité dépendant bien évidemment du nombre d'itérations choisi).

2. Simulation d'une distribution suivant un histogramme donné

2.1. Simulation suivant un exemple précis

Le but de cette simulation est de reproduire la répartition donnée suivant un histogramme connu. Cet exercice se découpe en deux parties. Tout d'abord nous allons reproduire la distribution non uniforme selon un histogramme à trois catégories, représentant le nombre d'individus des espèces A B et C. L'histogramme donné est celui-ci :

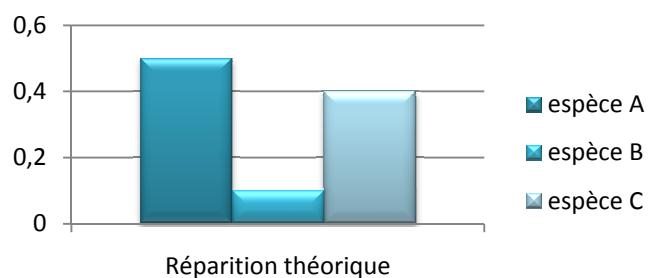


Figure 1 : Histogramme de répartition des espèces selon leurs probabilités

Ainsi, un individu tiré au hasard a une probabilité de 0,5 d'appartenir à l'espèce A, de 0,1 d'appartenir à l'espèce B et de 0,4 d'appartenir à l'espèce C.

La technique pour simuler une répartition suivant un tel histogramme est de le représenter sous la forme d'une table, qui cumule les répartitions, comme suit :

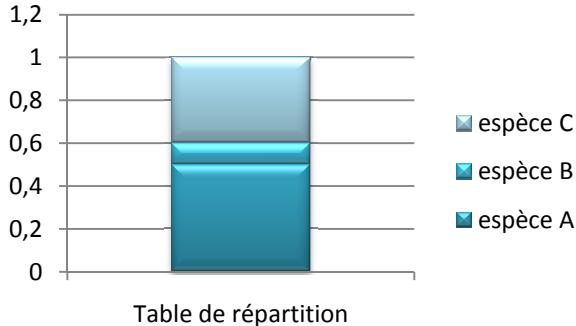


Figure 2 : Table de répartition des espèces

De cette façon, pour déterminer si l'individu tiré appartient à l'espèce A, B ou C, il suffit de le représenter par une variable X prenant ses valeurs entre 0 et 1. Ainsi, si la valeur de X est inférieure à 0,5, alors l'individu appartient à l'espèce A, si elle est comprise entre 0,5 et 0,6, alors il appartient à l'espèce B, et enfin si elle est supérieure à 0,6, alors il appartient à l'espèce C.

Nous avons testé ce programme sur 1000 tirages, puis sur 10 000, toujours avec le générateur de base rand.

**** Simulation histogramme ****

```
Nombre d'iterations : 1000
Especie A : 53.20
Especie B : 9.00
Especie C : 37.80
```

Que voulez vous faire ?
1 : Recommencer.
0 : Quitter.

**** Simulation histogramme ****

```
Nombre d'iterations : 10000
Especie A : 50.16
Especie B : 9.97
Especie C : 39.87
```

Que voulez vous faire ?
1 : Recommencer.
0 : Quitter.

On peut voir que la répartition observée suit de très près la répartition donnée par l'histogramme. Bien évidemment, les résultats sont meilleurs lorsque le nombre d'itérations devient conséquent.

Nous avons testé ce programme plusieurs, et voici les résultats moyens que l'on peut obtenir avec 10 simulations, et 10 000 tirages à chaque simulation :

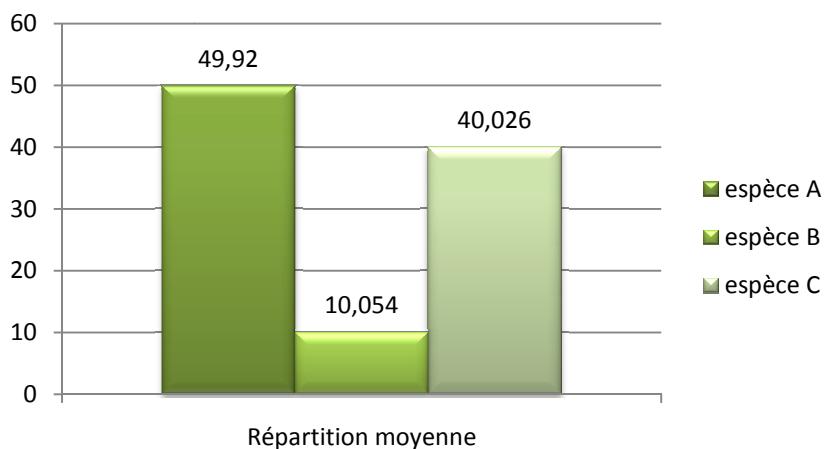


Figure 3 : Répartition moyenne sur 10 simulations à 10 000 tirages, suivant l'histogramme donné en exemple

2.2. Simulation suivant un histogramme quelconque

La seconde partie de cet exercice consiste en une généralisation de la méthode précédente : on fournit à l'utilisateur un histogramme de taille quelconque, et chaque classe contient un nombre d'individus quelconques. Il faut donc, à partir de cet histogramme, créer une table de répartition, et normaliser les tirages entre 0 et 1.

L'algorithme suivi est le suivant (le code pourra être trouvé en annexe) :

```

simulationTableau (tabeau de réels T, entier taille)
{
    Déclaration des variables nécessaires
    Initialisation du germe de rand
    Calcul de l'effectif total du tableau T

    Initialisation du tableau d'occurrences tab à 0
    Initialisation de l'histogramme :
    hist[0] ← T[0]
    pour i = 1..taille
        hist[i] ← hist[i-1] + T[i]

    Normalisation entre 0 et 1 :
    pour i = 0..taille
        hist[i] ← hist[i] / total

    Tirage des valeurs et remplissage du tableau des occurrences
    Pour cpt = 0..nbIter
    {
        Tirage d'un nombre aléatoire entre 0 et 1
        Incrémentation dans la case correspondante de tab
    }

    Affichage des résultats
}

```

Nous avons testé cette simulation avec un tableau quelconque, par exemple $T = \{27, 42, 53, 13, 7, 25\}$, et un nombre d'itérations de 1000, puis 10 000.

```
**** Simulation histogramme ****
```

```
Quelle taille de tableau : 6
Remplissage du tableau :
T[0] : 27
T[1] : 42
T[2] : 53
T[3] : 13
T[4] : 7
T[5] : 25
Nombre d'iterations : 1000
```

```
Affichage de l'histogramme
Hist[0] : 0.161677
Hist[1] : 0.413174
Hist[2] : 0.730539
Hist[3] : 0.808383
Hist[4] : 0.850299
Hist[5] : 1.000000
```

```
Affichage des resultats
Classe 0 : 168
Classe 1 : 237
Classe 2 : 321
Classe 3 : 66
Classe 4 : 57
Classe 5 : 151
```

```
**** Simulation histogramme ****
```

```
Quelle taille de tableau : 4
Remplissage du tableau :
T[0] : 56
T[1] : 78
T[2] : 24
T[3] : 32
Nombre d'iterations : 10000
```

```
Affichage de l'histogramme
Hist[0] : 0.294737
Hist[1] : 0.705263
Hist[2] : 0.831579
Hist[3] : 1.000000
```

```
Affichage des resultats
Classe 0 : 2944
Classe 1 : 4133
Classe 2 : 1317
Classe 3 : 1606
```

Si on teste 10 fois ces programmes, et qu'on représente sous forme graphique les histogrammes donnés au départ, et les histogrammes obtenus après simulation, on obtient les résultats suivants :

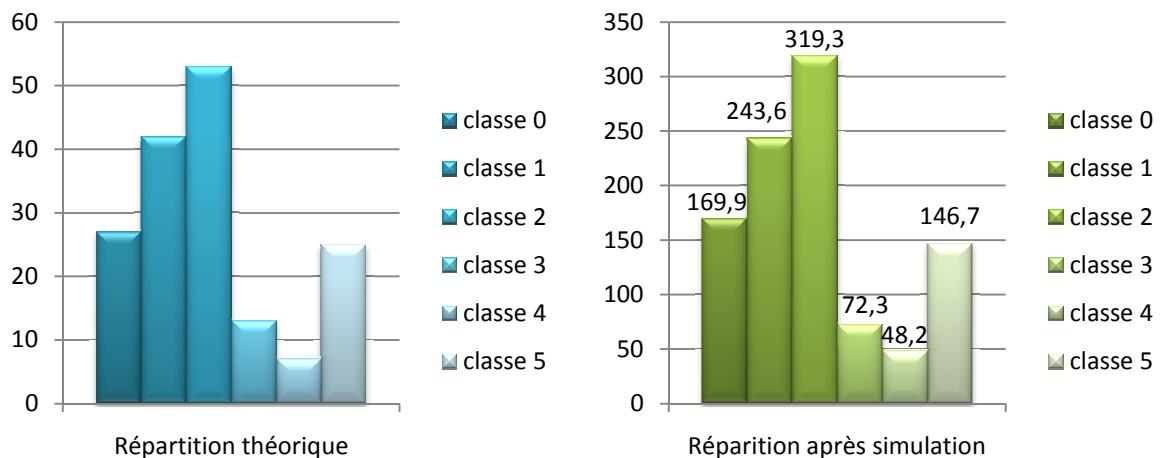


Figure 4 : Répartitions théorique et pratique moyenne selon un histogramme à 6 classes, sur 1000 itérations, sur 10 répétitions de l'expérience

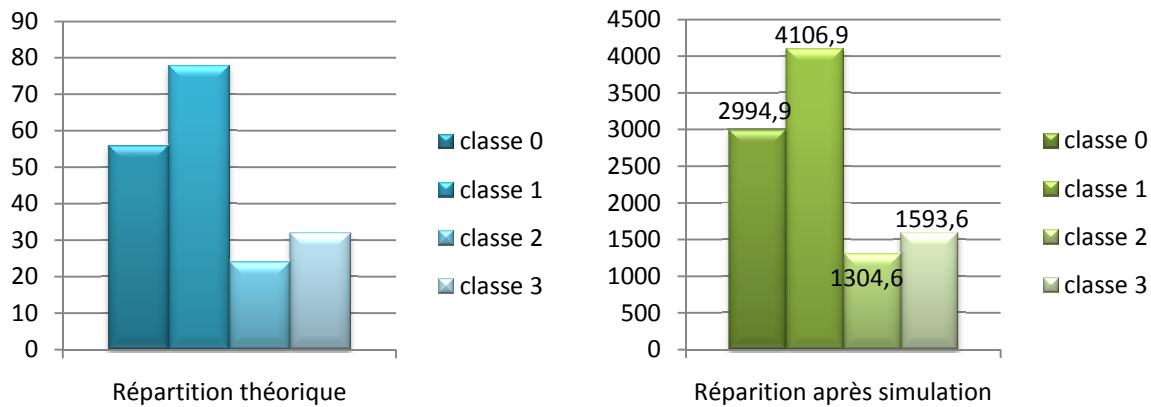


Figure 5 : Répartitions théorique et pratique moyenne selon un histogramme à 4 classes, sur 10 000 itérations, sur 10 répétitions de l'expérience

Il est clair que, comme pour l'exemple avec seulement 3 classes, la simulation d'une distribution suivant un histogramme donne des résultats uniformes (dépendants du nombre d'itérations choisi) avec le générateur rand, et ce quelque soit le nombre de classes de l'histogramme, ou l'effectif de chaque classe.

3. Simulation d'une loi continue inversible : la loi exponentielle

Une technique pour générer une distribution continue est la génération par loi inverse. Ainsi, si on considère que le nombre tiré représente $F(X)$, alors pour retrouver X il suffit d'appliquer l'inverse de F (si cette application existe) sur le nombre tiré. De cette façon, on utilise la forme explicite de l'inverse de sa fonction de répartition F , aussi appelée fonction quantile, que nous connaissons de manière explicite pour la loi exponentielle :

$$x = F^{-1}(\text{nombre tiré}) = -\text{Moyenne} * \ln(1 - \text{nombre tiré})$$

C'est cette formule que nous avons utilisé pour simuler la distribution selon une loi exponentielle négative. Dans cet exercice, nous avons considéré une moyenne théorique égale à 10. Nous avons simulé 1000 et 10 000 tirages, et avons écrit chacune des valeurs calculées grâce à la formule précédente dans un fichier. Nous avons ensuite traitées ces données avec Microsoft Office Excel (tri et représentation graphique), et les résultats obtenus sont les suivants :

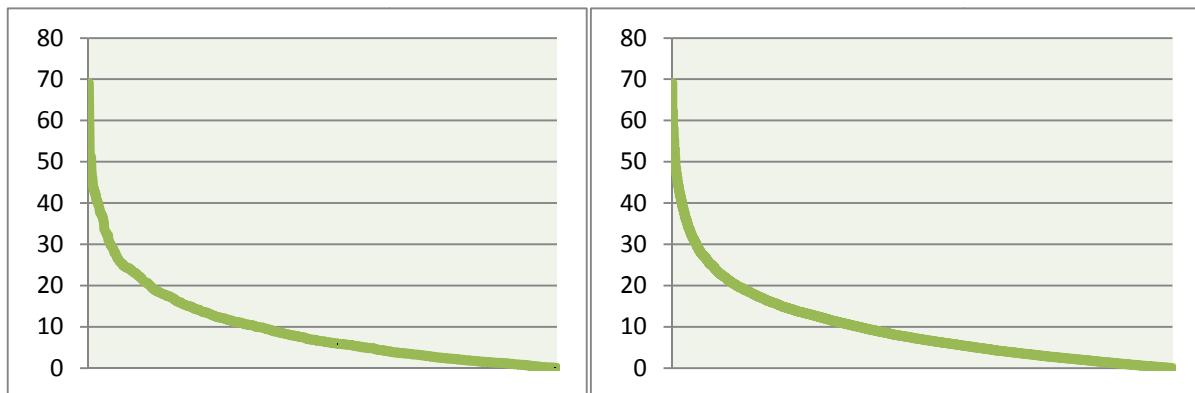


Figure 6 : Simulation de la fonction quantile d'une loi exponentielle de moyenne 10, sur 1000 et 10 000 itérations

Si l'on représente le résultat de la simulation à 1000 itérations sous forme d'un graphique donnant les fréquences d'apparition des valeurs, nous obtenons le graphique suivant :

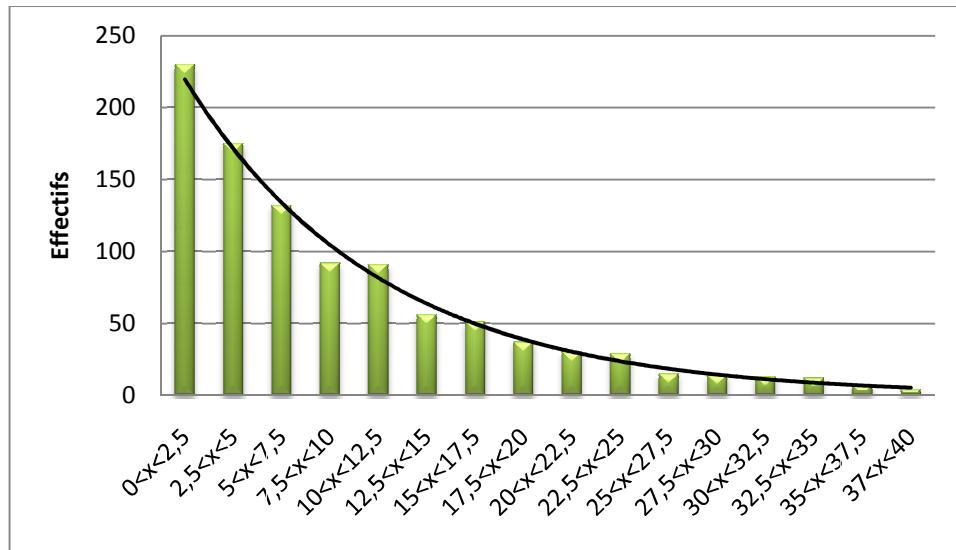


Figure 7 : Représentation graphique d'une simulation d'une répartition suivant une loi exponentielle négative, de moyenne 10, sur 1000 itérations

Si l'on exécute ce programme 10 fois, sur 10 000 itérations, et que l'on affiche la répartition selon les fréquences d'apparition des valeurs, nous obtenons le graphique suivant :

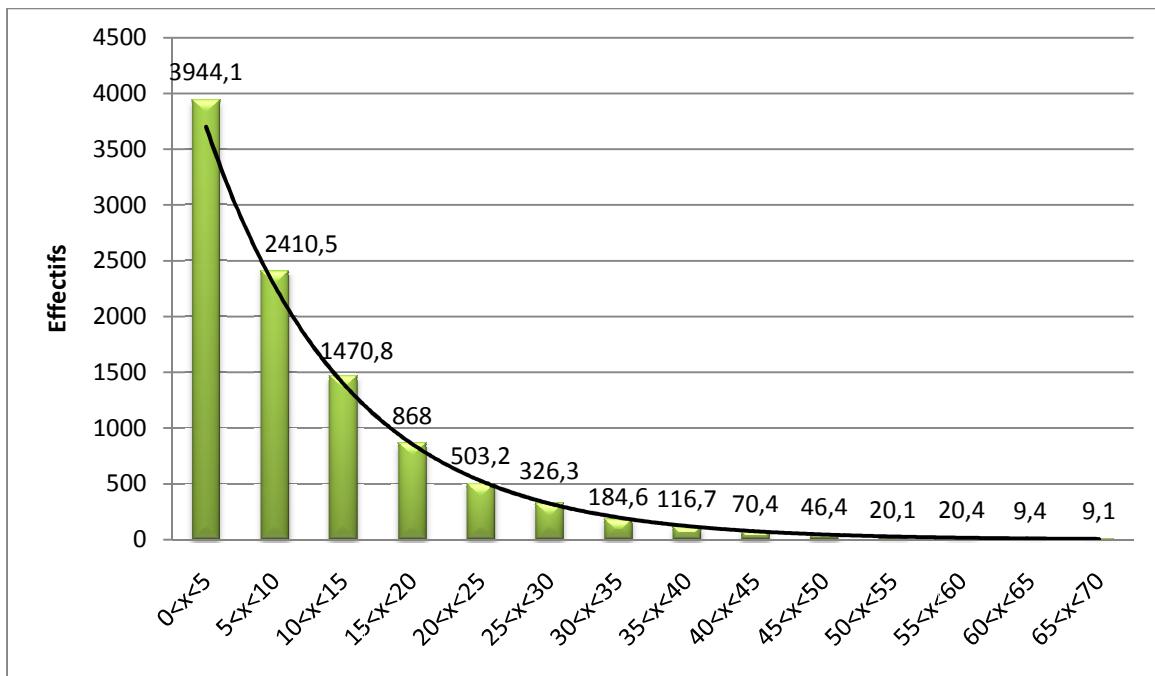


Figure 8 : Représentation graphique d'une simulation moyenne sur 10 expériences d'une loi exponentielle négative, de moyenne 10, sur 10 000 itérations à chaque expérience

4. Simulation d'une loi continue non inversible : la loi normale

4.1. Méthode de Box et Muller

Pour simuler une loi continue non inversible, une méthode très connue est la méthode de réjection. Cette méthode peut s'appliquer pour la simulation de la loi normale, mais il faut considérer des éléments supplémentaires, tel qu'un intervalle borné pour la fonction de densité. Il existe d'autres méthodes, plus particulièrement dédiée à la simulation d'une loi normale. La première méthode mise en œuvre est basée sur le théorème central limite. Cependant, cette technique est une méthode approchée, très dépendante du nombre de tirages que l'on effectue, et qui n'est pas d'une grande précision. On lui préférera la méthode développée par Box et Muller, qui repose sur le tirage de deux nombres pseudo-aléatoires (suivant une loi de répartition uniforme), puis sur le calcul de deux nombres indépendants, suivant une loi normale. Ce calcul repose sur les fonctions sinus et cosinus, ce qui le rend un peu plus lent, mais la méthode est exacte et bien plus précise.

Pour cet exercice, nous avons simulé 2000 tirages (2×1000 calculs) puis 20 000 (2×10000 calculs), répartis équitablement de chaque côté de la moyenne, que nous avons inscrit dans un fichier. En triant les valeurs et en les représentant graphiquement, nous obtenons la courbe ci-dessous, qui est bien la fonction quantile d'une loi normale.

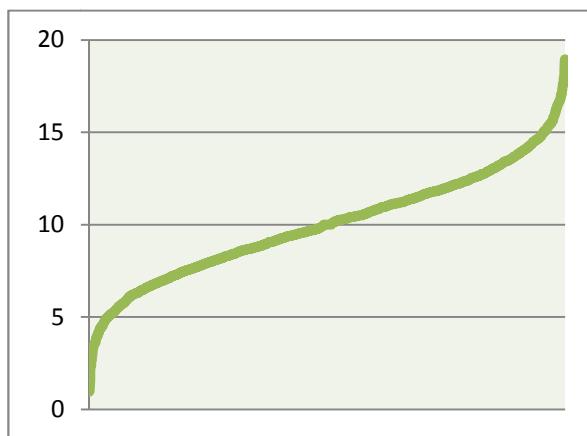


Figure 9 : Simulation de la fonction quantile d'une loi normale de moyenne 10 et d'écart-type 3, sur 1000 tirages, suivant la méthode de Box et Müller

Si l'on représente ce même exemple selon les fréquences d'apparition, nous obtenons le graphique suivant :

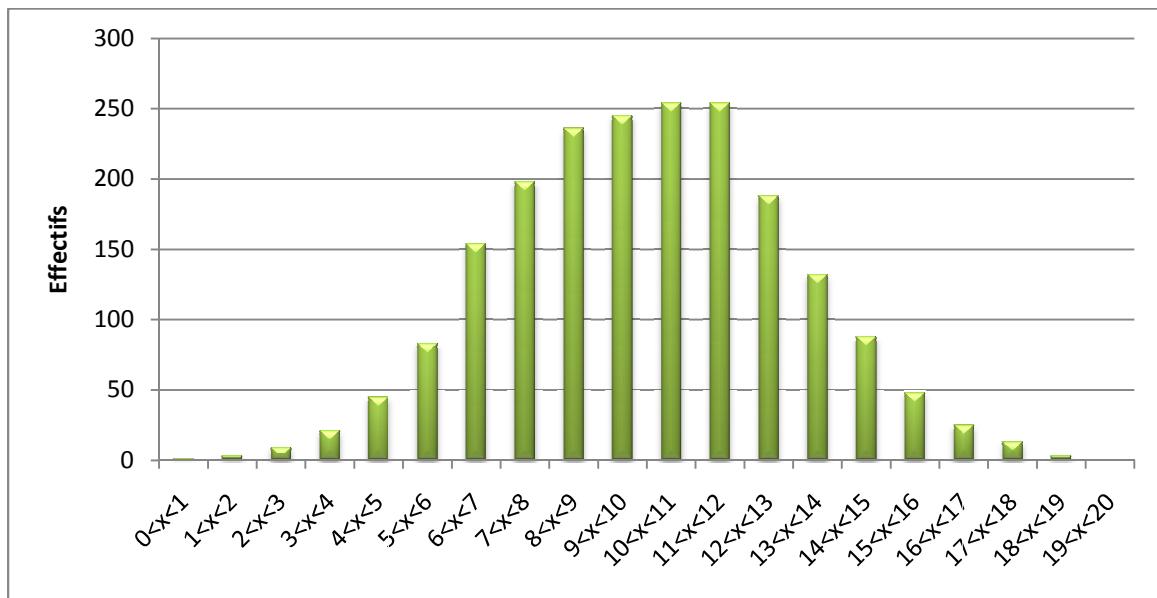


Figure 10 : Représentation graphique d'une simulation d'une distribution suivant une loi normale de moyenne 10 et d'écart-type 3

Si l'on exécute ce programme 10 fois, en utilisant 10 000 itérations à chaque expérience pour simuler une loi normale, nous obtenons la représentation graphique suivante :

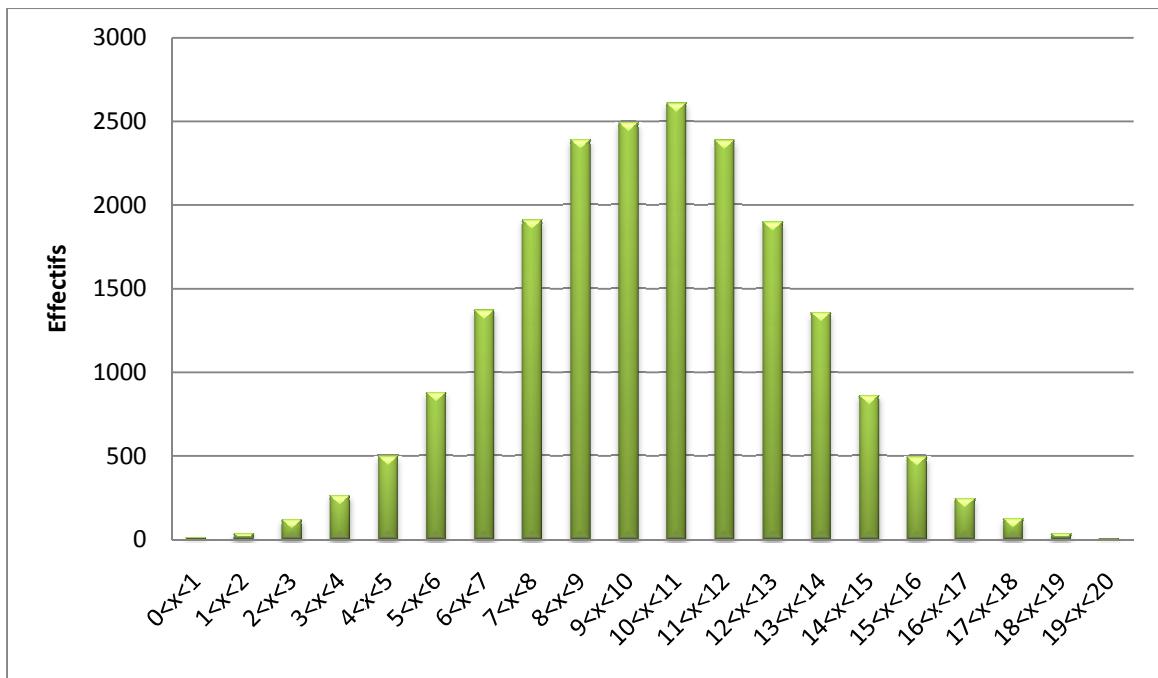


Figure 11 : Représentation graphique d'une simulation moyenne sur 10 expériences d'une loi normale, de moyenne 10 et d'écart-type 3, sur 10 000 itérations à chaque expérience

4.2. Autres algorithmes

Méthode de réjection adaptée

D'après le support de cours, il existe deux méthodes adaptées de la méthode de réjection pour la loi normale. Nous avons implémenté la première de ces deux méthodes, la seconde étant adaptée pour une simulation d'une loi normale centrée réduite (ce qui n'est pas notre cas ici) et l'avons testée avec 1000 et 10 000 tirages. Les résultats obtenus sont les suivants :

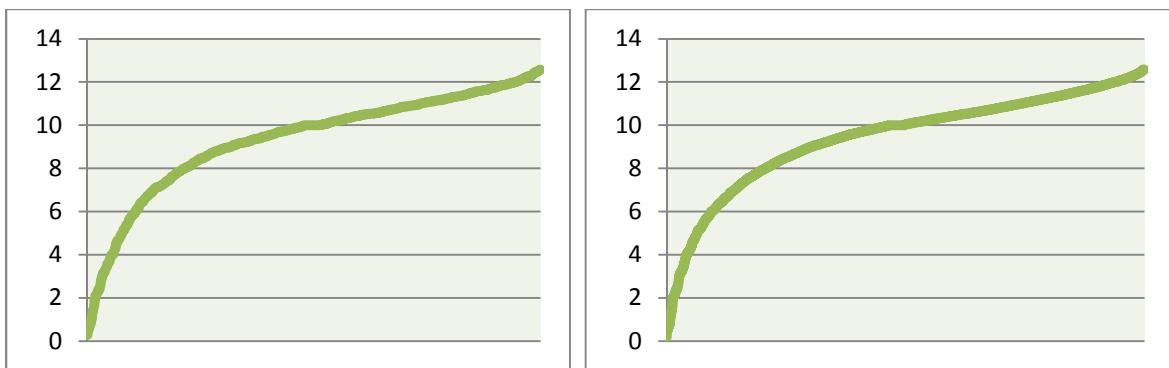


Figure 12 : Simulation de la fonction quantile d'une loi normale de moyenne 10 et d'écart-type 3, sur 1000 et 10 000 tirages, suivant la première méthode de réjection adaptée à la loi normale

Si l'on représente graphiquement les fréquences d'apparition des valeurs, sur 10 expériences, nous obtenons les résultats suivants, pour des simulations sur 1000 et 10 000 tirages.

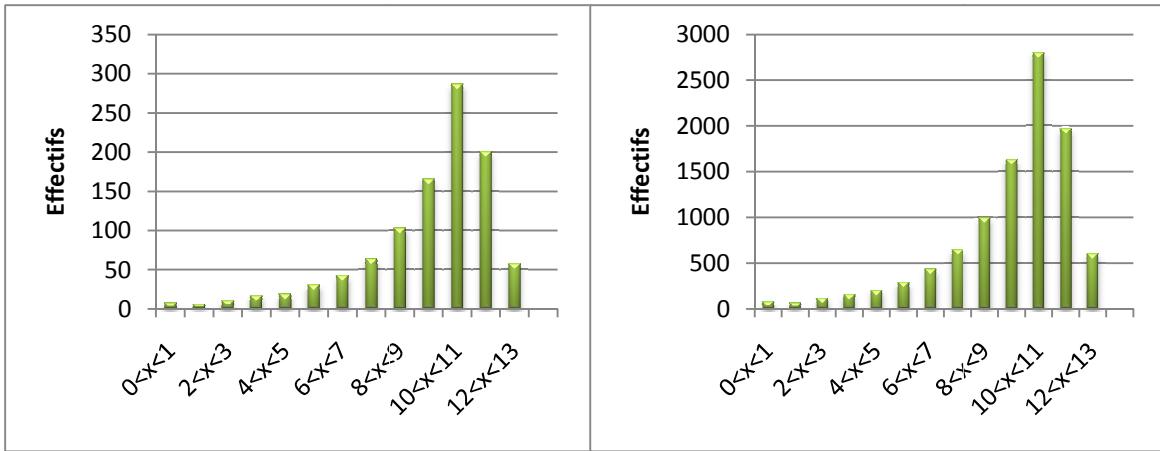


Figure 13 : Représentation graphique de deux simulations moyennes sur 10 expériences d'une loi normale de moyenne 10 et d'écart-type 3, respectivement suivant 1000 et 10 000 itérations par expérience, avec la première méthode de réjection adaptée

Comme on peut le voir, cette méthode simule bien toute la première partie de la « cloche » formée par une loi normale, mais retombe très rapidement pour la seconde partie. Cette méthode n'est donc pas forcément la plus appropriée pour simuler une loi normale.

Il existe d'autres méthodes permettant de simuler une loi normale. En effet, on trouve parmi les plus connue la méthode de Wallace, appelée aussi méthode par récurrence, ainsi que la méthode de Ziggurat. Cette dernière est utilisée pour simuler la loi normale sous Matlab. Elle doit son nom aux pyramides de Ziggurat : elle est basée sur une répartition en rectangles horizontaux, suivant la courbe de la loi normale, exception faite de la « première couche », c'est-à-dire la zone la plus proche de l'axe des abscisses. La méthode de Wallace quant à elle, repose sur des multiplications matricielles entre vecteurs gaussiens (vecteurs créés à partir de variables gaussiennes indépendantes) et une matrice orthogonale.

4.3. Bibliothèques de génération de nombres pseudo-aléatoires

Il existe plusieurs bibliothèques permettant de générer des nombres pseudo-aléatoires, la plupart suivant des répartitions uniformes. Cependant il est possible de les utiliser pour simuler des répartitions non uniformes. On peut compter parmi elles :

- Hasard : langage C (utilise des générateurs de nombres pseudo-aléatoires tels que : Mersenne Twister, ISAAC, Park-Miller, RANDU,...)
- CLIB : langage C
- Boost Random : langage C++
- The Scalable Parallel Random Number Generators Library (SPRNG) : langage C++
- Whrandom : Python
- GSL : langage C et C++

REFERENCES BIBLIOGRAPHIQUES

- [1] Fonction quantile, <http://www.math-info.univ-paris5.fr/smel/cours/mp/node15.html>
- [2] Génération de nombres pseudo-aléatoires suivant une distribution non uniforme par circuits intégrés programmables, Tarek Ould Bachir, <http://www.scribd.com/doc/5297290/Generation-de-nombres-pseudoaleatoires>. Date de publication : 2008.
- [3] Introduction à la simulation, Application à la simulation aléatoire à évènements discrets, David Hill, support de cours.
- [4] Loi exponentielle, http://fr.wikipedia.org/wiki/Loi_exponentielle
- [5] Vers des générateurs de nombres aléatoires uniformes et gaussiens à très haut débit, Renaud Santoro, http://tel.archives-ouvertes.fr/docs/00/43/86/00/PDF/these_santoro.pdf. Date de publication : 2009.
- [6] Sortie de la bibliothèque Hasard version 0.2, <http://linuxfr.org/~haypo/26716.html>. Date de publication : 31 mai 2008.
- [7] Bibliothèque Clib, www.bien-programmer.fr/clib.htm.
- [8] Boost random, http://www.boost.org/doc/libs/1_44_0/doc/html/boost_random.html.
- [9] SPRNG: Scalable Parallel Pseudo Random Number Generators Library, <http://sprng.cs.fsu.edu/>.
- [10] Wrandom, pseudo random number generator, <http://docs.python.org/release/2.4/lib/module-wrandom.html>.
- [11] GSL, GNU Scientific Library, <http://www.gnu.org/software/gsl/>.

ANNEXES

TABLE DES ANNEXES

Annexe 1 : Code source de l'exercice préliminaire, en langage C.....	II
Annexe 2 : Code source de l'exercice de simulation suivant un histogramme donné en exemple, en langage C. Le code pourra être modifié pour enregistrer les résultats dans des fichiers	II
Annexe 3 : Code source de l'exercice de simulation suivant un histogramme quelconque, en langage C. Le code pourra être modifié pour enregistrer les résultats dans des fichiers.....	III
Annexe 4 : Code source d'une simulation suivant une loi exponentielle, en langage C	IV
Annexe 5 : Code source d'une simulation suivant une loi normale par la méthode de Box et Müller, en langage C.....	V
Annexe 6 : Code source d'une simulation suivant une loi normale par une méthode de Réjection adaptée, en langage C	VI

Annexe 1: Code source de l'exercice préliminaire, en langage C.

```
void simulationUniforme (int A, int B, int iter)
{
    int i;
    int val;
    int * tab;

    /* On alloue autant de cases qu'il y a d'entiers entre A et B */
    tab = (int *) malloc(sizeof(int) * (B - A));
    for ( i = 0 ; i < (B-A) ; i++ )
        tab[i] = 0;

    if (tab != NULL)
    {
        for( i = 0 ; i < iter ; i++ )
        {
            val = rand() % (B - A) + A;
            tab[val - A]++;
        }

        for(i = 0; i < (B - A); i++)
            printf("Entre %d et %d\t: %d \n", i+A, i+A+1 ,tab[i]);

        free(tab);
    }
}
```

Annexe 2: Code source de l'exercice de simulation suivant un histogramme donné en exemple, en langage C. Le code pourra être modifié pour enregistrer les résultats dans des fichiers.

```
void simulationHistogramme(int nb)
{
    int i;
    int tab[3] = { 0 , 0 , 0 };
    srand(time(NULL));

    for( i = 0 ; i < nb ; i++ )
    {
        int var = rand() % 100 + 1;
        if( var <= 50 )
            tab[0]++;
        else if( var <= 60)
            tab[1]++;
        else
            tab[2]++;
    }

    printf("Especie A : %.2f\n", (float)tab[0]*100/nb);
    printf("Especie B : %.2f\n", (float)tab[1]*100/nb);
    printf("Especie C : %.2f\n", (float)tab[2]*100/nb);
}
```

Annexe 3 : Code source de l'exercice de simulation suivant un histogramme quelconque, en langage C. Le code pourra être modifié pour enregistrer les résultats dans des fichiers.

```
void simulationTableau (double *T, int taille, int nb)
{
    int      i, cpt, boolean = 0;
    int      *tab = (int *) malloc (sizeof(int) * taille);
    double   *hist = (double *) malloc (sizeof(double) * taille);
                /* hist = table de répartition */
    double   var, total = 0.0; /* contiendra le nombre total d'effectif */

    srand(time(NULL));

    /* Calcul de total */
    for ( i = 0 ; i < taille ; i++ )
        total += T[i];

    /* Remplissage du tableau d'occurrences */
    for ( i = 0 ; i < taille ; i++ )
        tab[i] = 0;

    /* Initialisation de l'histogramme */
    hist[0] = T[0];
    for ( i = 1 ; i < taille ; i++ )
        hist[i] = hist[i-1] + T[i];

    for ( i = 0 ; i < taille ; i++ )
        hist[i] = ((double) hist[i]) / ((double) total);

    /* Affichage de l'histogramme */
    for ( i = 0 ; i < taille ; i++ )
        printf("Hist[%d] : %lf\n", i, hist[i]);

    /* Tirage des valeurs et remplissage du tableau des occurrences */
    for ( cpt = 0 ; cpt < nb ; cpt ++ )
    {
        var = simuSimple(0,1);
        boolean = 0;
        i = 0;
        while ((boolean == 0)&&(i < taille))
        {
            if( var > hist[i] )
                i++;
            else
            {
                tab[i]++;
                boolean = 1;
            }
        }
    }

    /* Affichage des résultats */
    printf("\nAffichage des résultats\n");
    for ( i = 0 ; i < taille ; i++ )
        printf("Classe %d : %d\n", i, tab[i]);

    /* Libération de la mémoire */
    free(hist);
    free(tab);
}
```

Annexe 4 : Code source d'une simulation suivant une loi exponentielle, en langage C.

```
void expNeg (double moyenne, int nb)
{
    int      i, var;
    double   x;
    FILE    * fic;
    char     nom[100];

    srand(time(NULL));

    printf("Loi Exponentielle : Nom du fichier dans lequel enregistrer les
donnees : ");
    scanf("%s", nom);

    fic = fopen(nom, "w");
    if( fic == NULL )
        exit(-1);

    for( i = 0 ; i < nb ; i++)
    {
        /* Génération d'un nombre entre 1 et 1000 */
        var = rand() % 999 + 1;
        x = -moyenne * log( (1000.0 - (double)var) / 1000);
        /* Enregistrement des résultats dans un fichier */
        fprintf(fic, "%lf\n", x);
    }

    fclose(fic);
}
```

Annexe 5 : Code source d'une simulation d'une loi normale par la méthode de Box et Müller, en langage C.

```
void normaleBoxMuller(double moy, double ecart_t, int nbIter)
{
    double    x1, x2, y1, y2;
    FILE      * fic;
    char      nom[100];
    int       i;

    printf("Loi Normale : Nom du fichier dans lequel enregistrer les donnees
: ");
    scanf("%s", nom);

    fic = fopen(nom, "w");
    if( fic == NULL )
        exit(-1);

    srand(time(NULL));

    for ( i = 0 ; i < nbIter ; i++)
    {
        /* Génération des deux nombres aleatoires entre 0 et 1 */
        do{
            x1 = simuSimple(0,1);
        }while(x1==0);

        do{
            x2 = simuSimple(0,1);
        }while(x2==0);

        /* Calcul de deux nombres indépendants suivant la forme de Box et
        Muller */
        y1 = ( cos(2.0*PI * x2) ) * pow( ( (-2.0) * log (x1) ), 0.5);
        y2 = ( sin(2.0*PI * x2) ) * pow( ( (-2.0) * log (x1) ), 0.5);

        /* On rééquilibre pour avoir un loi centrée réduite */
        y1 *= ecart_t;
        y2 *= ecart_t;
        y1 += moy;
        y2 += moy;

        /* Ecriture des resultats dans un fichier */
        fprintf(fic, "%lf\t %lf\n", y1, y2);
    }

    fclose(fic);
}
```

Annexe 6 : Code source d'une simulation suivant une loi normale par une méthode de réjection adaptée, en langage C.

```
void normaleRejection1 (double moy, double ecart_t, int nbIter)
{
    double    a, b, c;
    double    r, x;
    FILE      * fic;
    char      nom[100];
    int       i;

    printf("Loi Normale rejection 1 : Nom du fichier dans lequel enregistrer
les donnees : ");
    scanf("%s", nom);

    fic = fopen(nom, "w");
    if( fic == NULL )
        exit(-1);

    srand(time(NULL));

    for( i = 0 ; i < nbIter ; i++)
    {
        do
        {
            /* Génération des deux nombres aleatoires entre 0 et 1 */
            a = simuSimple(0,1);
            b = simuSimple(0,1);
            c = a + b;
        } while ( ( c > 1 ) || ( a == 0 ) );

        /* Calcul du nombre suivant la loi normale */
        r = sqrt(-2*log(c)/a);
        x = moy + (a - b) * r * ecart_t;

        if( x < 0 )
        {
            i--;
            continue;
        }

        /* Enregistrement des résultats dans un fichier */
        fprintf(fic, "%lf\n", x);
    }
    fclose(fic);
}
```