TP2 - Simulation

Rémi CHASSAGNOL

October 13, 2022

Contents

1 Test de Mercenne Twister

Tout d'abord, il faut tester la répétabilité du générateur. L'archive téléchargée sur le site de **Makoto Matsumoto** contient un fichier `mt19937ar.out` qui correspond à la sortie attendue du programme. Ici, il suffit de tester si le programme affiche bien le résultat attendu. Pour ce faire, on redirige la sortie standard du programme vers un nouveau fichier, puis on utilise la commande `diff` pour tester si la sortie est correcte.

```
$ gcc mt19937ar.c
$ ./a.out > tmp.txt
$ diff tmp.txt mt19937ar.out
```

La commande `diff` n'affiche rien, la sortie du programme correspond bien à celle attendue.

2 Génération uniforme entre A et B

Pour générer un nombre aléatoire dans un intervalle $[a, b]$, nous allons utiliser la fonction `genrand_real1()` qui génère un nombre aléatoire entre 0 et 1. On obtient la fonction suivante:

```
/* G n re un nombre al atoire compris entre `a` et `b`. */
double uniformAB(double a, double b) {
  return a + (b - a)*genrand_real1();
}
```

Si on essais de générer **1000** nombres entre **-89.2** et **56.7** on obtient la répartition suivante:

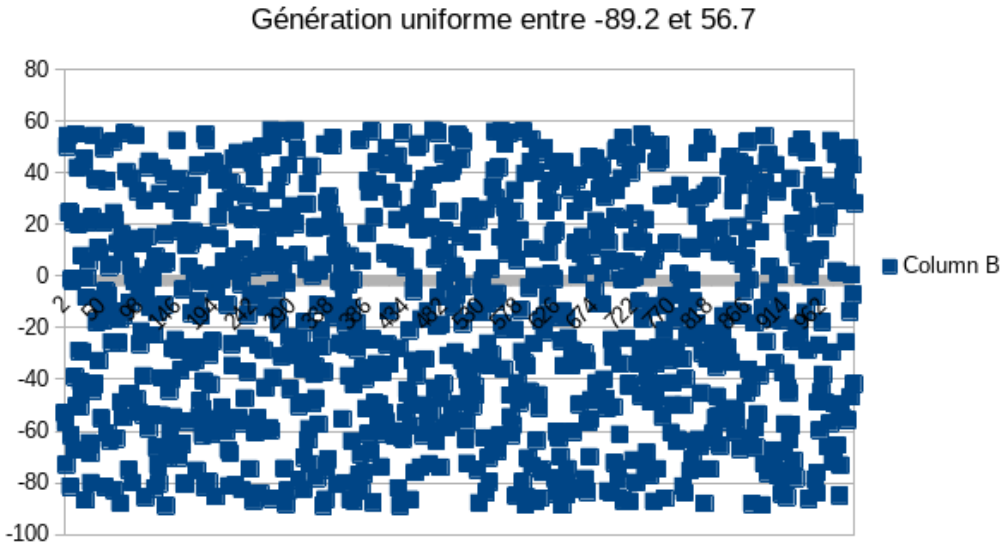

Figure 1: Répartition génération uniforme

3 Reproduction d'une distribution discrète empirique

3.1 Cas spécifique à 3 classes

Tout d'abord, on souhaite générer reproduire une distribution à partir de 3 classes A, B, et C. On dispose des informations suivantes:

- 350 observations de la classe A **(35%)**
- 450 dans la classe B **(45%)**
- 200 dans la classe c **(20%)**

Pour ce faire, on utilise la fonction `genrand_real1()` et le principe suivant:

- si elle retourne un nombre inférieur à `0.35`, on est dans la classe **A** (on retourne **0**)
- pour un nombre compris entre `0.35` et `0.80` on est dans la classe **B** (on retourne **1**)
- sinon on est dans la classe **C** (on retourne **2**)

Avec cette technique et sachant que `genrand_real1()` retourne des nombre pseudo-aléatoires uniformément répartit s dans $[0, 1]$, on a bien 35% de chance de tirer un individus de la classe A, 45% de la classe B et 20% de la classe C.

Le code source de la fonction est le suivant:

```
/* Reproduction d'une distribution discr te empirique avec les 3 classes A, B, C
 */
int genrand_discEmpDist() {
  double r = genrand_real1();
  int output = 2; // classe C par d faut
```

```
  if (r <= 0.35) { // classe A
    output = 0;
  } else if (r <= 0.80) { // classe B
    output = 1;
  }
  return output;
}
```

A noter que pour éviter une condition inutile, on se place dans la classe C par défaut et on change en si besoin.

On teste ensuite ce générateur en faisant plusieurs tirages que l'on comptabilise dans un tableau, puis on calcule le pourcentage d'individus trouvés dans chaque classes.

```
discEmpDist - 1000
A: 0.3430000000
B: 0.4550000000
C: 0.2020000000

discEmpDist - 10000
A: 0.3545000000
B: 0.4447000000
C: 0.2008000000

discEmpDist - 100000
A: 0.3493800000
B: 0.4503200000
C: 0.2003000000
```

On observe déjà de bons résultats pour 1000 tirages, et on peut effectivement voir que la précision augmente avec le nombre de tirages. On passe de près de 2% d'erreur sur la classe A pour 1000 à 0.2% pour 100 000 tirages.

3.2 Fonction plus générique

Ici, on suit le même principe que précédemment sauf que le but est d'implémenter une fonction qui prend en compte un nombre indéterminé de classes. Cette fonction prend en entrée le nombre de classe et un tableau d'effectifs, qui correspond aux effectifs que l'on souhaite observer pour chaque classes. On commence par calculer l'effectif total puis, à l'aide d'une boucle `for`, on test pour toutes les classes, si l'effectif cumulé divisé par l'effectif total est supérieur ou égal à un nombre aléatoire `r` tiré à l'aide de `genrand_real1()`.

```
int genrand_discEmpDist2(int nbClasses, int effectifs[]) {
  double r = genrand_real1();
  int output = 2;
  int effectifTotal = 0;
  int effectifCum = 0;
  int i;

  // calcul effectif total
  for (i = 0; i < nbClasses; ++i){
    effectifTotal += effectifs[i];
  }
```

```
  // calcul du resultat
  for (i = 0; i < nbClasses; ++i){
    effectifCum += effectifs[i];
    if (r <= (double) effectifCum / effectifTotal) {
      output = i;
      i = nbClasses;
    }
  }
  return output;
}
```

On test la fonction avec 5 classes et des effectifs choisis arbitrairement et on obtient des résultats similaires à la précédente question.

Pour `effectifs[5] = {20, 5, 50, 10, 15}`:

```
discEmpDist2 - 100000
0: 0.2026500000
1: 0.0494200000
2: 0.4980800000
3: 0.0993600000
4: 0.1504900000
```

4 Reproduction de distribution continues

Dans cette partie, on va générer des nombre aléatoires suivant une loi exponentielle négative de moyenne donnée. Pour ce faire on utilise la technique d'anamorphose pour inverser la loi. On obtient donc la fonction suivante:

```
double negExp(double mean) {
  return -mean*log(1 - genrand_real2());
}
```