

TP 1 : Premiers programmes, structures de contrôle et simulations de variables aléatoires

Les programmes suivants sont des modèles à partir desquels vous pourrez écrire les programmes demandés plus bas.

```
1 #include <iostream> //bibliothèque de gestion des flux d'entrées et de
2 // sorties ("in and out stream")
4 using namespace std; // permet d'écrire "cout" et "cin" au lieu
5 // de "std::cout" et "std::cin"
6
7 int main() {
8     int n;
9     cout<<"En quelle annee es-tu ne ?"<<endl;
10    cin>>n;
11    cout<<"Donc tu as eu 20 ans en "<<n+20<<" , n\'est-ce pas ?"<<endl;
12    return 0;}
```

```
1 #include<iostream>
2 using namespace std;
4 double ajouter(double x, double y) {
5     return x+y;}
6
7 int main(){
8     double a,b;
9     cout<<"a ?"<<endl;
10    cin>>a;
11    cout<<"b ?"<<endl;
12    cin>>b;
13    cout<<ajouter(a,b)<<endl;
14    return 0;
15 }
```

```
1 #include <iostream>
2 #include <ctime> //gestion du temps, utile pour réinitialiser
3 // la graine de l'aléa avant la première utilisation
4 #include <cmath> //bibliothèque d'outil mathématiques
5
6 using namespace std;
7
8 inline double unif_rand() // "inline" s'utilise, optionnellement,
9 // pour les fonctions courtes
10 {return 2*(rand()+0.5)/(RAND_MAX+1.0)-1;}
11
12 int main() {
13     srand(time(0)); //on initialise la graine de rand
14     rand(); //on écoute la première itération de l'aléa, biaisée.
15     for ( int n = 0 ; n < 10 ; ++n )
16         cout << unif_rand() << endl;}
```

1 Recopier ces programmes (sans les parties de commentaires, situées à droite des //), les compiler et les exécuter.

Rappel : on appelle `nom_programme.cpp` un programme C++, on le compile de façon à en faire un exécutable avec la commande

```
> g++ nom_programme.cpp -o nom_executable
(attention, les > ne font pas partie des commandes!!), et on l'exécute ensuite avec la commande
> ./nom_executable
```

2 *Ecrire, sur le modèle du deuxième programme ci-dessus, une fonction de prototype*

```
double distribuer(double x, double y, int z)
```

*dont le résultat est  $(x+y)z$ . La tester dans un programme.*

3 *Que va afficher le programme suivant ?*

```
#include <iostream>
2 #include <cmath>
using namespace std;
4
int main() {
6     int n;
    cout<<1/3<<endl;
8     cout<<1/(double (3))<<endl;
    return 0;
10 }
```

4 *Ecrire un programme affichant sur la console une variable aléatoire de loi de Cauchy.*

Rappel : Pour  $U$  v.a. de loi uniforme sur  $[-\pi/2, \pi/2]$ ,  $\tan(U)$  est une variable aléatoire de loi de Cauchy (c'est la méthode de simulation par *inversion de la fonction de répartition*).

5 *Ecrire un programme demandant à l'utilisateur le choix de deux paramètres réels  $a, b$ , renvoie un message d'erreur si  $a \geq b$  et affiche sur la console une variable aléatoire de loi uniforme sur  $[a, b]$  sinon.*

6 *Ecrire un programme demandant à l'utilisateur le choix d'un paramètre réel  $\lambda$  et d'un entier  $n$  et affichant sur la console  $n$  simulations de variables aléatoires de loi exponentielle de paramètre  $\lambda$ .*

Rappel : Pour  $U$  v.a. de loi uniforme sur  $[0, 1]$ ,  $-\log(U)/\lambda$  est une variable aléatoire de loi exponentielle de paramètre  $\lambda$  (c'est la méthode de simulation par *inversion de la fonction de répartition*).

7 *Ecrire un programme demandant à l'utilisateur le choix d'un entier  $n$  et affichant sur la console  $n$  simulations de variables aléatoires de loi de Wigner, la loi de Wigner étant la loi de support  $[-2, 2]$  et de densité*

$$\frac{1}{2\pi} \sqrt{4 - x^2}.$$

Rappel : Pour simuler une v.a. de densité  $f$  de support  $[a, b]$  et telle que  $0 \leq f \leq M$ , on utilise une suite  $(U_n, V_n)_{n \geq 1}$  de couples indépendants de variables aléatoires telles que pour tout  $n$ ,  $U_n, V_n$  sont indépendantes et de lois uniformes sur respectivement  $[a, b]$ ,  $[0, M]$ , on pose

$$\tau = \min\{n \geq 1; V_n \leq f(U_n)\},$$

et la variables aléatoire  $U_\tau$  suit alors la loi de densité  $f$ . C'est la méthode de *simulation par rejet*. Notons que plus  $M$  est petit, plus cette méthode est rapide, on a donc intérêt à choisir  $M = \|f\|_\infty$ .

```

#include <iostream>
2 #include <fstream>//permet de gerer les lectures/écritures sur
    // les fichiers extérieurs
4
using namespace std;
6
int main() {
8     ofstream flux("nom_fichier.dat"); //on crée un flux de sortie
        // nommé "flux", qui
10        // écrit, non pas sur la console comme "cout", mais sur
        // le fichier nom_fichier.dat,
12        // qui est créé, dans le répertoire ambiant,
        // s'il n'existe pas, et écrasé puis
14        // recréé s'il existe déjà.
    flux<<34<<endl;//on écrit 34 sur la première ligne
16        // du fichier nom_fichier.dat
    flux<<56<<endl;//on écrit 56 sur la deuxième ligne
18        // du fichier nom_fichier.dat
    flux<<77<<endl;//on écrit 77 sur la troisième ligne
20        // du fichier nom_fichier.dat
    flux.close();}

```

8 Recopier le programme ci dessus, le compiler et l'exécuter. Ouvrir ensuite le fichier `nom_fichier.dat` avec un éditeur de texte.

9 Ecrire un programme demandant à l'utilisateur le choix d'un entier  $n$  et créant un fichier `echantillon_wigner.dat` sur lequel il écrit, en colonne,  $n$  simulations de variables aléatoires de loi de Wigner. Ouvrir ensuite Scilab, et afficher l'histogramme de cet échantillon (pour  $n = 1500$ ), que l'on compare à la densité, avec les commandes suivantes (à écrire dans la fenêtre Scilab).

```

> (on se rend sur le répertoire ambiant avec les commandes Linux)
> echant=read("echantillon_wigner.dat",-1,1);
> abscisses=-2:0.01:2;
> ordonnées=(2*%pi)^(-1)*sqrt(4-abscisses.*abscisses);
> histplot(50, echant);
> plot2d(abscisses,ordonnées,5);

```

10 On définit, pour  $a$  entier strictement positif, la suite  $(u_n)$  par  $u_0 = a$  et

$$u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{sinon.} \end{cases}$$

C'est une conjecture admise que le nombre

$$N_a = \inf\{n; u_n = 1\}$$

(dépendant bien entendu de  $a$ ) est alors fini, quel que soit  $a$ .

1) Ecrire un programme qui demande à l'utilisateur de taper  $a$  et affiche toutes les valeurs de  $u_n$  pour  $n \leq N_a$ , ainsi que  $N_a$ .

2) Ecrire un programme qui demande à l'utilisateur de taper  $A$  et affiche

$$\max\{N_1, \dots, N_A\}.$$

11 A rendre, par mail, en binôme, à [florent.benaych@gmail.com](mailto:florent.benaych@gmail.com). Sujet du mail : C++IFMA, Prénom 1 NOM 1, Prénom 2 NOM 2.

Ecrire un programme demandant à l'utilisateur le choix de l'affichage sur la console ou de l'écriture sur un fichier et d'un entier  $n$  et qui réalise les instructions suivantes :

- si le choix est l'écriture sur la console, on simule et affiche sur la console  $\min(n, 20)$  réalisations indépendantes de la loi Gaussienne standard,

- si le choix est l'écriture sur un fichier, on simule et écrit sur un fichier `echant_gauss.dat`, en colonnes,  $n$  réalisations indépendantes de la loi Gaussienne standard.

On utilisera la méthode de simulation de Box-Muller, décrite ci-dessous, en essayant, si possible, d'utiliser des variables `static` pour n'avoir à simuler les variables aléatoires  $U$  et  $\theta$  que  $n/2$  fois. On comparera avec la densité Gaussienne (qui vaut  $\frac{1}{\sqrt{2\pi}} \exp(-\frac{x^2}{2})$  en tout  $x$  de  $\mathbb{R}$ ) standard via Scialb.

Rappel : Pour  $U, \theta$  variables aléatoires de loi uniforme sur respectivement  $[0, 1]$ ,  $[0, 2\pi]$ , les variables parties réelles et imaginaires du nombre complexe  $\sqrt{-2\log(U)}e^{i\theta}$  sont deux variables aléatoires indépendantes de loi Gaussienne standard.

**12** *Ecrire un programme qui demande à l'utilisateur de saisir un entier  $N$  et qui affiche le  $N$ -ième nombre premier.*

**13** *Ecrire un programme qui demande à l'utilisateur de saisir deux entiers  $n, m$  et qui affiche leur pgcd.*

Rappel : L'algorithme d'Euclide permet de calculer le pgcd  $n \wedge m$  de deux nombres  $n, m$  suivant les règles suivantes :

$$\begin{cases} n \wedge m = m & \text{si } n = 0, \\ n \wedge m = m \wedge n & \text{pour tous } n, m, \\ n \wedge m = n \wedge r & \text{où } r \text{ est le reste de la division euclidienne de } m \text{ par } n, \text{ pour tous } n \leq m. \end{cases}$$