RAPPORT DU TP
‘PROJETS INFORMATIQUES’

TP°2
‘Generation of Random Variates’

Eldis YMERAJ- GA- #22015179

Année universitaire 2021-2022

Exercice 1 : “Dernière implémentation C” du Mersenne Twister (MT) original.

Dans cette exercice je vais tester si le code de ‘Mersenne Twister / 2002 version’ marche bien et quel est le but de ce code. Dans les prochaines exercices je vais avoir besoin pour les random fonctions qui sont écrit dans ce code pour réaliser le travaille demandé.

Dans le main les fonctions genrand_int32() et gen rand_real12()n son appelées et ces fonctions permettent de afficher les nombres aléatoires dans intervales [0,1) et l’autre dans (0,1). Il est demandé la liste des 1000 prmiérs.

```c
int main(void)
{
    int i;
    unsigned long init[4]={0x123, 0x234, 0x345, 0x456}, length=4;
    init_by_array(init, length);
    printf("1000 outputs of genrand_int32()\n");
    for (i=0; i<1000; i++) {
      printf("%10lu ", genrand_int32());
      if (i%5==4) printf("\n");
    }
    printf("\n1000 outputs of genrand_real2()\n");
    for (i=0; i<1000; i++) {
      printf("%10.8f ", genrand_real2());
      if (i%5==4) printf("\n");
    }
    return 0;
}
```

J’ai copié le code que je l’ai trouvé dans “Matsumoto Home Page” et le résultat après le compilation est :

```
1000 outputs of genrand_real2()
0.76275443 0.99000644 0.98670464 0.10143112 0.27933125
0.69867227 0.94218740 0.03427201 0.78842173 0.28180608
0.92179002 0.20785655 0.54534773 0.69644020 0.38107718
0.23978165 0.65286910 0.07514568 0.22765211 0.94872929
0.74557914 0.62664415 0.54708246 0.90959343 0.42043116
0.86334511 0.19189126 0.14718544 0.70259889 0.63426346
0.77408121 0.04531601 0.04605807 0.88595519 0.69398270
0.05377184 0.61711170 0.05565708 0.10133577 0.41500776
0.91810699 0.22320679 0.23353705 0.92871862 0.98897234
0.19786706 0.80558809 0.06961067 0.55840445 0.90479405
0.63288060 0.95009721 0.54948447 0.20645042 0.45000959
0.87050869 0.70806991 0.19406895 0.79286390 0.49332866
0.78483914 0.75145146 0.12341941 0.42030252 0.16728160
0.59906494 0.37575460 0.97815160 0.39815952 0.43595080
0.04952478 0.33917805 0.76509902 0.61034321 0.90654701
0.92915732 0.85365931 0.18812377 0.65913428 0.28814566
0.59476081 0.27835931 0.60722542 0.68310435 0.69387186
0.03699800 0.65897714 0.17527003 0.02889304 0.86777366
0.12352068 0.91439461 0.32022990 0.44445731 0.34903686
0.74639273 0.65918367 0.92492794 0.31872642 0.77749724
0.85413832 0.76385624 0.32744211 0.91326300 0.27458185
0.22190155 0.19865383 0.31227402 0.85321225 0.84243342
0.78544200 0.71854080 0.92503892 0.82703064 0.88306297
0.47284073 0.70059042 0.48003761 0.38671694 0.60465770
0.41747204 0.47163243 0.72750808 0.65830223 0.10955369
0.64215401 0.23456345 0.95944940 0.72822249 0.40888451
0.69980355 0.26677428 0.57333635 0.39791582 0.85377858
0.76962816 0.72004885 0.90903087 0.51376506 0.37732665
0.12691640 0.71249738 0.81217908 0.37037313 0.32772374
```

```
1000 outputs of genrand_int32()
1067595299  955945823  477289528 4107218783 4228976476
3344332714 3355579695  227628506  810200273 2591290167
2560260675 3242736208  646746669 1479517882 4245472273
1143372638 3863670494 3221021970 1773610557 1138697238
1421897700 1269916527 2859934041 1764463362 3874892047
3965319921   72549643 2383988930 2600218693 3237492380
2792901476  725331109  605841842  271258942  715137098
3297999536 1322965544 4229579109 1395091102 3735697720
2101727825 3730287744 2950434330 1661921839 2895579582
2370511479 1004092106 2247096681 2111242379 3237345263
4082424759  219785033 2454039889 3709582971  835606218
2411949883 2735205030  756421180 2175209704 1873865952
2762534237 4161807854 3351099340  181129879 3269891896
 776029799 2218161979 3001745796 1866825872 2133627728
  34862734 1191934573 3102311354 2916517763 1012402762
2184831317 4257399449 2899497138 3818095062 3030756734
1282161629  420003642 2326421477 2741455717 1278020671
3744179621  271777016 2626330018 2560563991 3055977700
4233527566 1228397661 3595579322 1077915006 2395931898
1851927286 3013683506 1999971931 3006888962 1049781534
1488758959 3491776230  104418065 2448267297 3075614115
3872332600  891912190 3936547759 2269180963 2633455084
1047636807 2604612377 2709305729 1952216715  207593580
2849898034  670771757 2210471108  467711165  263046873
3569667915 1042291111 3863517079 1464270005 2758321352
3790799816 2301278724 3106281430    7974801 2792461636
 555991332  621766759 1322453093  853629228  686962251
1455120532  957753161 1802033300 1021534190 3486047311
1902128914 3701138056 4176424663 1795608698  560858864
3737752754 3141170998 1553553385 3367807274  711546358
2475125503  262969859  251416325 2980076994 1806565895
 969527843 3529327173 2736343040 2987196734 1649016367
2206175811 3048174801 3662503553 3138851612 2660143804
```

Exercice 2:

Dans cette exercice j'ai créé un fonction 'uniform' qui a deux nombre réel a et b comme paramètres et génére pseudo random nombres entre a et b. Dans notre cas on va essayer avec a=-89. 2°C et b=56.7°C, donc on va afficher tout les températures entre -89. 2°C et 56.7°C.

Ici on a déclaré un fonction uniform de type double qui prends pour paramètre deux nombres réel a et b et qui retourne la formule vue en cours. Et dans le main on a mis les valeurs de a et b et on est entré dans un boucle pour, pour afficher les 10 premièrs températures entre -89. 2°C et 56.7°C.

```c
double uniform(double a, double b){
    return a+(b-a)*genrand_real1();
}
int main(void)
{
    double a,b;
    a=-89.2;
    b=56.7;
    printf("For temperatures between -89,2 °C and 56,7 °C we have :\n");
    for(int i=0;i<10;i++){
        printf("%f °C\n", uniform(a,b));
    }
    return 0;
}
```

Quad on compile on va avoir :

```
For temperatures between -89,2 °C and 56,7 °C we have :
29.668187 °C
-69.433905 °C
42.955043 °C
32.627753 °C
-70.672624 °C
52.157808 °C
44.061537 °C
-56.951133 °C
3.061215 °C
-44.238427 °C
```

Exercice 3:

A) On a des données de terrain avec 3 classes d'où, 500 observations en classe A, 150 en classe B et 350 en classe C. Le probabilité dans chaque classe est 50% pour A, 15% pour B et 35% pour C. Dans notre exo on va implémenter et tester un progra,,e simultant le distribution discrète qui est bien expliquée dans l'exercice avec les trois classes A, B et C.

```
int main(void)
{
    printf("Qestion 2-A : \n");
    double tab[]={0.5, 0.65, 1};
        int countA=0;
        int countB=0;
        int countC=0;
        for(int i=0; i<1000; i++){
    double RAND=genrand_real1();
       if(RAND<tab[0]){
            countA++;
       }else{
            if(RAND<tab[1]){
                countB++;
            }else{
                countC++;
            }
       }
    }
```

Premièrement j'ai déclaré un tableau de type double qui prends les valeur de probabilité pour chaque classe, tab[0] c'est pour la classe A , tab[1] c'est pour la classe B et tab[2] c'est pour la classe C. Aussi j'ai déclaré trois variables de type int countA, countB et countC initialisé à 0 qui sont des variables qu'ils comptent le nombre des fois ou un nouveau nombre aléatoire appartient dans la classe précisé. On est entrée dans un boucle for et on a déclaré un nouveau variable RAND de type double qui génére les nombres aléatoires entre 0 et 1 pour chaque itération. Pour chaque classe on a fait les conditions comme il est demandé dans l'exo et chque fois que la condition est vrai on fait countA/B/C++ pour compter les nombres dans chaque classe.

```
printf("%d times A\n", countA);
printf("%d times B\n", countB);
printf("%d times C\n", countC);
double pA=(countA*100/1000);
double pB=(countB*100/1000);
double pC=(countC*100/1000);
printf("Percentage obtained fo A is %.0f\n", pA);
printf("Percentage obtained fo B is %.0f \n", pB);
printf("Percentage obtained fo C is %.0f\n", pC);
return 0;
```

Pour les afficher, premiérement on a afficher combien de nombres on a dans chaque catégorie et après on a calculé le probabilité pour 1000 itérations. Pour 1000000 itérations je l'ai mis en

commentaire car c'est la meme chose mais juste on change nombre des itérations et on divise par 1000000.

P.s : pA, pB, pC sont des probabilités pour chaque classe, ils sont des nouvelles variables de type double.

Quand on compile on a:

```
Qestion 2-A for 1000 iterations:
517 times A
141 times B
342 times C
Percentage obtained for A is 51
Percentage obtained for B is 14
Percentage obtained for C is 34
```

```
Qestion 2-A for 1000000 iterations:
500111 times A
150442 times B
349447 times C
Percentage obtained for A is 50
Percentage obtained for B is 15
Percentage obtained for C is 34
```

B) Pour la question b, j'ai créé un fonction qui prends pour paramétre un tableau du type double et sa taille n. J'ai déclaré un autre tableau nommée arr de type int initialisé à 0. Maintement on est entré dans un autre boucle for et on va générer 1000 random numbers. On a déclaré un variable real de type double qui va nous géneré pour chaque itération un nouveau nobre aléatoire réel. On met le condition pour chaque classe, si le nombre est inférieure à 0.5 on est dans classe A, s'il est inférieure à 0.65 on est dans classe B et sinon on est dans C et chaqu fois que le condition est vrai pour chaque classe on les compter. Pour l'affichage j'ai fait un autre boucle for et j'ai calculé la probabilité et après je les ai affiché. Dans le main j'ai expliqué que classe 1 correspond à A, classe 2 correspond à B et classe 3 correspond à C. J'ai initialisé le tableau tab avec les probabilitées correspondants et

après j'ai fait appel à function qu'on a créé. Pour 1'00'0000 itérations c'est la meme chose mais on juste change le nombre d'itérations dans le boucle.

```c
double fonction(double tab1[], int n){
    int arr[n];
    for(int i=0;i<n;i++){
        arr[i]=0;
    }
    for(int i=0;i<1000;i++){
        double real=genrand_real1();
        for(int j=0;j<n;j++){
            if(real<tab1[j]){
                arr[j]++;break;
            }
        }
    }
    for(int i=0;i<n;i++){
        arr[i]=(arr[i]*100)/1000;
        printf("Percentage of class %d is %d\n", i+1,arr[i]);
    }
}
int main(void)
{
    printf("Distribution with 1000 drawings: \n");
    printf("Class 1 corresponds to class A\n");
    printf("Class 2 corresponds to class B\n");
    printf("Class 3 corresponds to class C\n");
    printf("===============Distribution with 1000 drawings ===============\n");
    //printf("===============Distribution with 1000 drawings ===============\n");
    double tab[3]={0.5, 0.65,1};
    fonction(tab,3);

    return 0;
}
```

```
Class 1 corresponds to class A
Class 2 corresponds to class B
Class 3 corresponds to class C
===============Distribution with 1000 drawings ================
Percentage of class 1 is 51
Percentage of class 2 is 14
Percentage of class 3 is 34
```

```
Class 1 corresponds to class A
Class 2 corresponds to class B
Class 3 corresponds to class C
===============Distribution with 1000000 drawings ================
Percentage of class 1 is 50
Percentage of class 2 is 15
Percentage of class 3 is 34
```

Exercice 4:

```
double negExp(double Mean){
    double Y=genrand_real1();
    double X=log(1-Y);
    return ((-Mean)*X);

}
```

Premièrement, dans cette exercice j'ai crée un fonction quil s'appelle negExp et qui prend pour paramétre un 'mean' de type double. Ce fonction permet de générer le loi exponentiel négatif. J'ai appelé Y un variable de type double qui permet de génerer un nombre réel aleatoire entre dans intervalle 0 ,1. X est une variable de type double qui calcule une piece de la formule donné au cours et à la fin on retourne la for,ule complet.

Maintement dans le main j'ai testé le foncton negExp pour 1'000 et 1'000'000 nombres génerés et je les ai mis dans un tableau. J'ai appelé arr1 le tableau qui contient les 1'000 nombres génerés de notre fonction negExp et arr2 pour 1'000'000 nombres génerés. Sum1 et sum2 sont deux variabes qu'on a besoin de calculer la moyenne pour chaque tableau et avg1 et avg2 sont deux variables qu'ils vont contenir les valeurs de moyennes calculées pour 1'000 et 1'000'000 nombres génerés par notre fonction. On est entré dans un boucle for pour mettre dans le tableau les 1'000 nombres génerés par le fonction negExp et on a fait la meme chose pour la deuxième tableau avec 1'000'000 itérations. Maintement on a calculé la somme des élements dans chque tableau et après la moyenne pour arr1 et arr2.

```
int main(void)
{
    double arr1[1000]; //An arry for 1'000 drawings
    double arr2[1000000]; //An arry for 1'000'000 drawings
    double sum1=0; // Is a variable which we need to calculate the avarage for 1'000 drawings
    double sum2=0; // Is a variable which we need to calculate the avarage for 1'000'000 drawings
    double avg1; //Is a variable wich will calculate the avarage for 1'000 drawings
    double avg2; //Is a variable wich will calculate the avarage for 1'000'000 drawings
    for(int i=0;i<1000;i++){
    arr1[i]=negExp(11);
}
    for(int i=0;i<1000000;i++){
    arr2[i]=negExp(11);
}
    for(int i=0;i<1000;i++){
        sum1+=arr1[i];
    }
    for(int i=0;i<1000000;i++){
        sum2+=arr2[i];
    }
    avg1=sum1/1000;
    avg2=sum2/1000000;
    printf("Average obtained after drawing 1'000 is: %f\n", avg1);
    printf("Average obtained after drawing 1'000'000 is: %f", avg2);

    return 0;

}
```

La moyenne dans les deux cas est proche de 11. Voila ce que est affiché dans la console.

```
Average obtained after drawing 1'000 is: 10.865714
Average obtained after drawing 1'000'000 is: 10.995488
```

La dernière question de l'exercice 4 nous demande de tester la la fréquence des nombres entre 0 et 1, entre 1 et 2,… et entre 20 et 21. Notre tableau qu'on va créé joue le role d'un tableau poubelle et au dessous de 21 on prends juste une poubelle et pour chaque numéro tiré on compte dans quelle case il apparait et on le calcule pour tous les tirages 1'000 et 1'000'000.

```
int testBin[22]; // An arry which test the frequency of numbers between 0 and 1
    for (int i=0;i<22;i++){
        testBin[i]=0;
    }
    for(int i=0;i<1000;i++){
        int BIN=((int) negExp(11)); // A variable which takes the value of fonc
        testBin[BIN]++;
    }
    printf("\n============  Distribution for 1000 drawings  ============\n");
    for(int i=0;i<22;i++){
        printf("%d  ",testBin[i]);
    }
    printf("\n===========  Distribution for 1000000 drawings  ===========\n");
    for(int i=0;i<1000000;i++){
        int BIN=((int) negExp(11));
        testBin[BIN]++;
    }
    for(int i=0;i<22;i++){
        printf("%d  ",testBin[i]);
    }
```

J'ai créé un tableau de type int avec 22 cases et je l'ai initialisé à 0. On est entré dans un boucle pour avec 1'000 tirages et j'ai déclaré u variable de type int que je l'ai nommé BIN qui prends des nombres genérés par notre fonction negExp(entiers) et après on les compte pour chaque itération. On a fait la meme chose pour 1'000'000 itérations.

Dans le console on a:

```
Average obtained after drawing 1'000 is: 10.865714
Average obtained after drawing 1'000'000 is: 10.995488

============  Distribution for 1000 drawings  =============
91  88  73  66  59  50  36  39  51  37  46  42  29  26  24  23  12  19  20  15  10  11
===========  Distribution for 1000000 drawings  ===========
86703  79745  72219  66009  60399  55054  50649  45906  42108  38144  35492  31733  29340  26687  24454
 22234  20568  18512  17040  15443  13956  12761
```

Exercice 6:

Des bibliothèques en C/C++ et Java qui génèrent des variables aléatoires :

- AMP RNG C++
- Easy Random JAVA
- GNU Scientific Library C/C++
- java.util.Random JAVA

CONCLUSION

Dans la première TP, j'ai mis en place différentes techniques pour la génération de nombres aléatoires mais ici j'ai utilisé l'un des meilleurs générateurs proposés pour la science au 21e siècle pour résoudre les exercices données. J'ai vu que l'utilisation des fonctions pour générer des nombres aléatoires n'est pas utile juste en Informatique et mathématique mais dans de nombreux domaines de la vie. Ce que j'ai vu dans ce TP est ce que les nombres aleatoires sont très important dans les calcules pour trouver probabilité et pour étudier differents domaines en science et aussi pour genérér et étudier les graphes pour le developpement de la science.

P.s : Dans ce rapport vous pouvez trouver les erreurs ortographiques car je trouve des difficultés dans la langu francaise, et les commentaires et les noms de variables dans le code je les ai écrit en Anglais. Je vous dis désolé pour ces types de fautes ortographiques et je vais essayer te travailler avec la langue pour ne pas faire ces erreurs.