Rapport TP Simulation

Génération de nombre pseudo-aléatoires suivant différentes distributions

Maxime ESCOURBIAC - Jean-Christophe SEPTIER
ZZ2 • F2 • 27 octobre 2010

But du TP:

Le but de ce tp est d'exploiter les résultats obtenues lors du précédent tp sur la génération de nombre aléatoire afin de simuler des lois de distribution réelles non uniformes.

On pourra décomposer cette étude avec e plan suivant:

- La reproduction de distributions discrètes.
  - Génération de nombres uniformément reparties entre A et B.
  - Génération d'un distribution selon un histogramme.
  - Fonction générale de distribution selon un histogramme.

- La reproduction de distributions continues.
  - La loi uniforme.
  - La loi exponentielle négative.
  - La loi normale avec la méthode de Box & Muller.
  - La loi normale selon la méthode de réjection.

- Recherche de library de génération de nombres pseudo-aléatoires suivant des lois de distributions non uniformes.

Maxime ESCOURBIAC • Adresse électronique : escourbi@poste.isima.fr • ISIMA
Jean-Christophe SEPTIER • Adresse électronique : septier@poste.isima.fr • ISIMA

Les distributions discrètes

Le générateur de nombres reparties entre A et B

La génération de ces nombres se base sur le générateur de nombre aléatoire rand()

Le code de la fonction:

```
int gene_nombre_AB(int min,int max)
{
    return(rand()%(max-min+1) +  min);
}
```

Relevé de mesures: (valeur min tirée 10, valeur max tirée 14) sur 1000 tirages

```
new-host-2:Desktop escourbi$ ./geneAB
nombre de 10 tires = 194
nombre de 11 tires = 221
nombre de 12 tires = 201
nombre de 13 tires = 188
nombre de 14 tires = 196
new-host-2:Desktop escourbi$ ./geneAB
nombre de 10 tires = 164
nombre de 11 tires = 203
nombre de 12 tires = 211
nombre de 13 tires = 216
nombre de 14 tires = 206
new-host-2:Desktop escourbi$ ./geneAB
nombre de 10 tires = 198
nombre de 11 tires = 197
nombre de 12 tires = 201
nombre de 13 tires = 190
nombre de 14 tires = 214
```

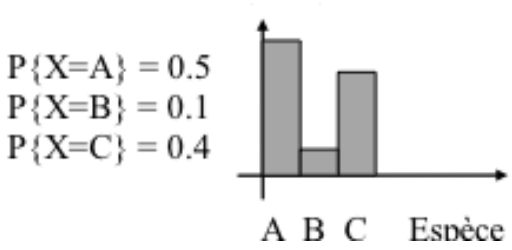
En conclusion, on pourra noter que le générateur utilisé est de mauvaise qualité.

En effet, sur le tirage 2 sur le nombre de 10 tirée, on obtient un écart de 18% par rapport au résultat théorique.

Les distributions discrètes

Générateur selon un histogramme.

La distribution devra se calquer sur l'histogramme suivant:

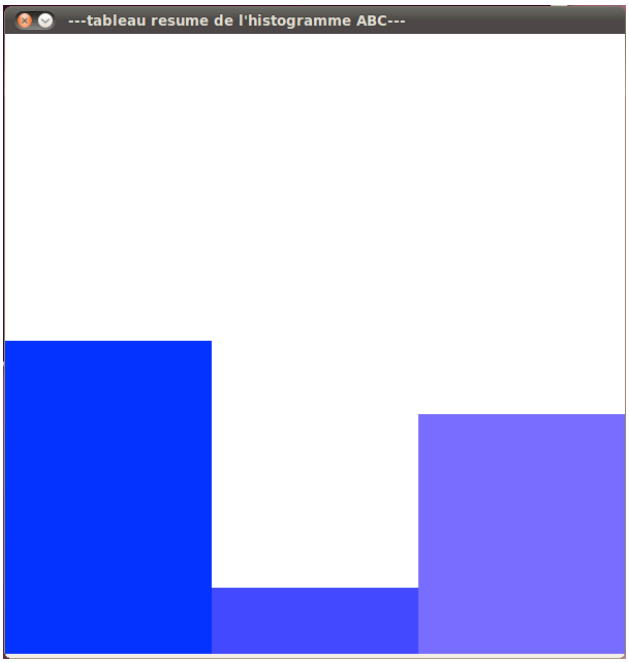


Le code de la fonction:

```
/**
 * \fn int * test_histogramme_ABC()
 * \brief simule le recencement d'especes
 * \return le tableau d'entier contenant le nombre d'elements de chaque espece
 */

int * test_histogramme_ABC()
{
  int i;
  float val;
  int * tab;
  tab = (int *) calloc(3,sizeof(int));
  if(!tab)
  {
     fprintf(stderr,"erreur alloc tab \n");
     exit(EXIT_FAILURE);
  }

  for(i=0;i<1000;i++)
  {
     val= gene_rand();
     if(val > 0.6)
        ++tab[2];
     else
     {
        if(val > 0.5)
           ++tab[1];
        else
           ++tab[0];
     }
  }
  return tab;
}
```

Les relevés de mesures:

Voici un aperçu d'un tirage réalisé avec la library SDL en C. (les proportions sont respectées)



et voici 5 relevés texte:

```
---tableau resume de l'histogramme ABC---
tab[0] = 50.500000 percent
tab[1] = 10.800000 percent
tab[2] = 38.700000 percent

---tableau resume de l'histogramme ABC---
tab[0] = 51.000000 percent
tab[1] = 10.700000 percent
tab[2] = 38.300000 percent

---tableau resume de l'histogramme ABC---
tab[0] = 51.200000 percent
tab[1] = 8.800000 percent
tab[2] = 40.000000 percent

---tableau resume de l'histogramme ABC---
tab[0] = 48.600000 percent
tab[1] = 10.000000 percent
tab[2] = 41.400000 percent

---tableau resume de l'histogramme ABC---
tab[0] = 52.000000 percent
tab[1] = 9.500000 percent
tab[2] = 38.500000 percent
```

Conclusion:

Les résultats correspondent bien à l'histogramme initial,

Les distributions discrètes

Fonction générale de distributions selon un histogramme.

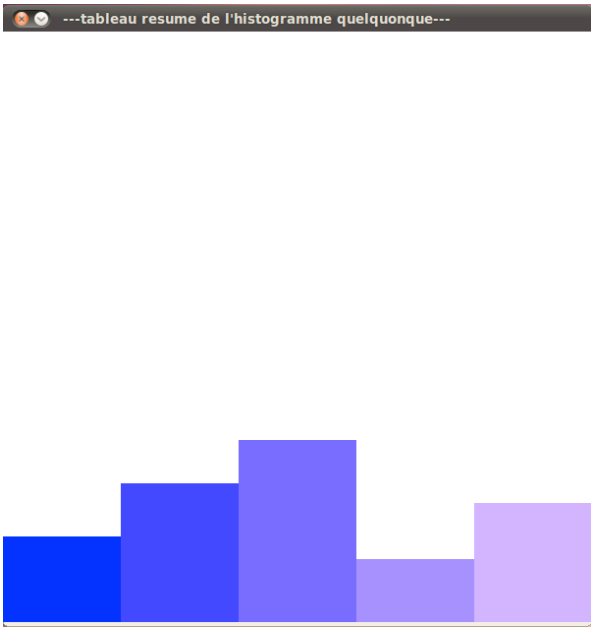
Code de la fonction:

```
/**
 *\fn int * histogramme_gene(int * data,int size, int effectif)
 *\brief fonction generale de distributions selon un histogramme donne
 *\param data tableau representant un l'effectif des classes
 *\param size nombre de classe
 *\param effectif nombre total d'element
 *\return tableau contenant le nombre de tirage par classe
 */

int * histogramme_gene(int * data,int size, int effectif)
{
    int i,j,*tab;
    float *histo,alea,cumul = 0.0;

    tab = (int *) calloc(size,sizeof(int));
    histo = (float *) calloc(size,sizeof(float));

    if(!tab || !histo)
    {
        fprintf(stderr,"erreur allocation tableau \n");
        exit(EXIT_FAILURE);
    }

    /*creation du tableau de repartition*/
    for(i=0;i<size;++i)
    {
       cumul += (float)data[i] / (float)effectif;
       histo[i] = cumul;
    }

    /*simulation avec le genrateur rand*/
    for(i=0;i<1000;++i)
    {
        alea = gene_rand();
        j = 0;
        while(j<size && alea > histo[j])
        {
            ++j;
        }
        if(j==size)
           tab[size-1]++;
        else
           tab[j]++;
    }

    free(histo);
    return tab;
}
```

Les relevés de mesures:

Effectifs initiaux: A: 15 , B=25 , C=30 , D =10 , E=20

Voici un aperçu d'un tirage réalisé avec la library SDL en C. (les proportions sont respectées)



et voici 3 relevés texte: (pour 1000 tirages)

```
new-host-2:Desktop escourbi$ ./gene
 espece 0 : 15.500000 percent
 espece 1 : 26.300000 percent
 espece 2 : 30.200000 percent
 espece 3 : 9.600000 percent
 espece 4 : 18.400000 percent
new-host-2:Desktop escourbi$ ./gene
 espece 0 : 14.300000 percent
 espece 1 : 26.700000 percent
 espece 2 : 29.600000 percent
 espece 3 : 10.300000 percent
 espece 4 : 19.100000 percent
new-host-2:Desktop escourbi$ ./gene
 espece 0 : 14.200000 percent
 espece 1 : 24.000000 percent
 espece 2 : 31.000000 percent
 espece 3 : 10.200000 percent
 espece 4 : 20.600000 percent
```

Conclusion:

Cette fonction permet de réaliser une distribution selon un tableau d'effectif mis en paramètre.

Les distributions continues

La loi uniforme selon la technique de l'anamorphose.

Le but de cette partie est de simuler une distribution suivant la loi continue entre 2 nombres, le principe est le même que pour la première partie, sauf que la valeur retournée est à valeur réelle.

code de la fonction :

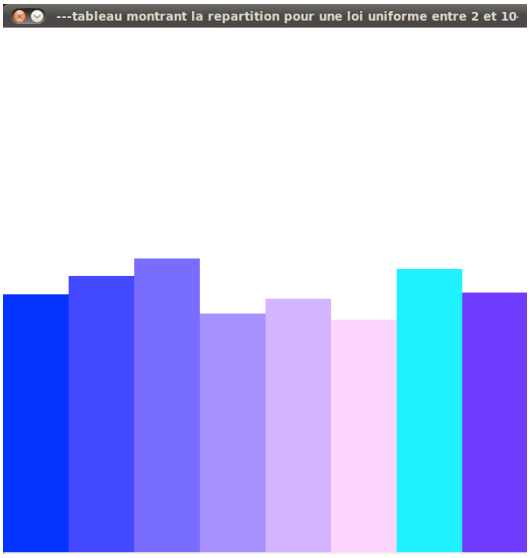
```
/**
 * \fn int * loi_uniforme(float a, float b)
 * \brief fonction qui effectue un tirage selon une loi uniforme entre a et b
 * \param a borne inferieure de la loi uniforme
 * \param b borne superieure de la loi uniforme
 * \return un tableau contenant le nombre de tirage effectue parmi des intervalles donn√©s
 *
 * Pour simuler ce fonctionnement, on utilise la forme explicite de la fonction inverse de la
 * fonction de repartition.
 * ici F^1(x) = A+(B-A)*x;
 */

int * loi_uniforme(double a, double b)
{
  /*dans cet exemple on prendra des intervalles de 1 */
  /*pour des questions de lisibilit√© du tableau, on testera que pour valeurs de
    a et b entieres */
  int i;
  int * tab;
  float val;

  tab = (int *) calloc((int)b - (int)a,sizeof(int));
  if(!tab)
  {
     fprintf(stderr,"erreur alloc tab \n");
     exit(EXIT_FAILURE);
  }
  for(i=0;i<1000;++i)
  {
     val = a+(b-a)*gene_rand();
     tab[(int)(val-a)]++;
  }
  return tab;
}
```

Relevé de mesure:

Pour représenter les mesures, on utilise un tableau où chaque case représente un intervalle.

Ici, les essais ont porté sur une simulation de loi continue ayant pour bornes (2,10)

Voici un aperçu d'un tirage réalisé avec la library SDL en C. (les proportions sont respectées)




```
---tableau montrant la repartition pour une loi uniforme entre 2.000000 et 10.000000---
 intervalle 2.000000 - 3.000000 = 123
 intervalle 3.000000 - 4.000000 = 132
 intervalle 4.000000 - 5.000000 = 140
 intervalle 5.000000 - 6.000000 = 114
 intervalle 6.000000 - 7.000000 = 121
 intervalle 7.000000 - 8.000000 = 111
 intervalle 8.000000 - 9.000000 = 135
 intervalle 9.000000 - 10.000000 = 124

---tableau montrant la repartition pour une loi uniforme entre 2.000000 et 10.000000---
 intervalle 2.000000 - 3.000000 = 110
 intervalle 3.000000 - 4.000000 = 127
 intervalle 4.000000 - 5.000000 = 130
 intervalle 5.000000 - 6.000000 = 123
 intervalle 6.000000 - 7.000000 = 133
 intervalle 7.000000 - 8.000000 = 128
 intervalle 8.000000 - 9.000000 = 129
 intervalle 9.000000 - 10.000000 = 119

---tableau montrant la repartition pour une loi uniforme entre 2.000000 et 10.000000---
 intervalle 2.000000 - 3.000000 = 129
 intervalle 3.000000 - 4.000000 = 109
 intervalle 4.000000 - 5.000000 = 126
 intervalle 5.000000 - 6.000000 = 128
 intervalle 6.000000 - 7.000000 = 135
 intervalle 7.000000 - 8.000000 = 128
 intervalle 8.000000 - 9.000000 = 124
 intervalle 9.000000 - 10.000000 = 120

---tableau montrant la repartition pour une loi uniforme entre 2.000000 et 10.000000---
 intervalle 2.000000 - 3.000000 = 126
 intervalle 3.000000 - 4.000000 = 130
 intervalle 4.000000 - 5.000000 = 111
 intervalle 5.000000 - 6.000000 = 129
 intervalle 6.000000 - 7.000000 = 127
 intervalle 7.000000 - 8.000000 = 132
 intervalle 8.000000 - 9.000000 = 131
 intervalle 9.000000 - 10.000000 = 112

---tableau montrant la repartition pour une loi uniforme entre 2.000000 et 10.000000---
 intervalle 2.000000 - 3.000000 = 137
 intervalle 3.000000 - 4.000000 = 123
 intervalle 4.000000 - 5.000000 = 122
 intervalle 5.000000 - 6.000000 = 116
 intervalle 6.000000 - 7.000000 = 128
 intervalle 7.000000 - 8.000000 = 131
 intervalle 8.000000 - 9.000000 = 118
 intervalle 9.000000 - 10.000000 = 124
```

Conclusion:

Même constat que pour la simulation de la loi continue discrète, on peut remarquer un décalage entre la théorie et la pratique.

Les distributions continues

La loi exponentielle négative selon la technique de l'anamorphose.

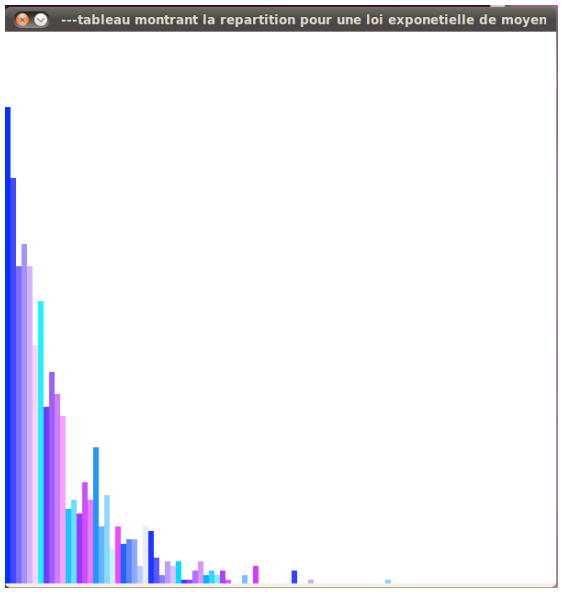
Cette partie est la même que la précédente sauf qu'on simule une loi exponentielle négative.

code de la fonction:

```c
 /**
 * \fn  int * loi_exponentielle_neg(float moyenne,float max)
 * \brief fonction qui effectue un tirage selon une loi exponentielle negative de moyenne M
 * \param moyenne moyenne de la loi exponentielle
 * \param max valeur maximale tolere
 * \return un tableau contenant le nombre de tirage effectue parmi des intervalles donn√©s
 *
 * Pour simuler ce fonctionnement, on utilise la forme explicite de la fonction inverse de la
 * fonction de repartition.
 * ici F^1(x) = -Moyenne*ln(1-valeur_tiree);
 */

int * loi_exponentielle_neg(double moyenne,double max)
{
   int i;
   int * tab;
   double val,moy = 0.0;

   tab = (int *) calloc((int)max,sizeof(int));
   if(!tab)
   {
      fprintf(stderr,"erreur alloc tab \n");
      exit(EXIT_FAILURE);
   }
   for(i=0;i<1000;++i)
   {
      do
      {
         val = -moyenne*log(1.0 - gene_rand());
      }while(isinf(val)); /*en cas si gene_rand = 1.0 => ln(0) indefini*/
      /*par contre isinf n'est pas une fonction ANSI C*/

      moy += val;
      if(val<max)
         tab[(int)(val)]++;
   }
   moy /= 1000.0;

   fprintf(stdout,"Moyenne reelle = %f \n",moy);

   return tab;
}
```

Relevé de mesure: les essais ont porté sur une loi exponentielle de moyenne 10

Chaque barre du graphe représente un intervalle de 1. (représentation du 1er tirage)




```
Moyenne reelle = 9.431970
 intervalle 0.000000 - 1.000000 = 108
 intervalle 1.000000 - 2.000000 = 92
 intervalle 2.000000 - 3.000000 = 72
 intervalle 3.000000 - 4.000000 = 77
 intervalle 4.000000 - 5.000000 = 72
 intervalle 5.000000 - 6.000000 = 54
 intervalle 6.000000 - 7.000000 = 64
 intervalle 7.000000 - 8.000000 = 40
 intervalle 8.000000 - 9.000000 = 48
 intervalle 9.000000 - 10.000000 = 43
 intervalle 10.000000 - 11.000000 = 38
 intervalle 11.000000 - 12.000000 = 17
 intervalle 12.000000 - 13.000000 = 19
 intervalle 13.000000 - 14.000000 = 16
 intervalle 14.000000 - 15.000000 = 23
 intervalle 15.000000 - 16.000000 = 19
 intervalle 16.000000 - 17.000000 = 31
 intervalle 17.000000 - 18.000000 = 13
 intervalle 18.000000 - 19.000000 = 20
 intervalle 19.000000 - 20.000000 = 8
```

```
Moyenne reelle = 9.986006
 intervalle 0.000000 - 1.000000 = 102
 intervalle 1.000000 - 2.000000 = 72
 intervalle 2.000000 - 3.000000 = 89
 intervalle 3.000000 - 4.000000 = 57
 intervalle 4.000000 - 5.000000 = 61
 intervalle 5.000000 - 6.000000 = 56
 intervalle 6.000000 - 7.000000 = 48
 intervalle 7.000000 - 8.000000 = 54
 intervalle 8.000000 - 9.000000 = 45
 intervalle 9.000000 - 10.000000 = 43
 intervalle 10.000000 - 11.000000 = 49
 intervalle 11.000000 - 12.000000 = 36
 intervalle 12.000000 - 13.000000 = 17
 intervalle 13.000000 - 14.000000 = 25
 intervalle 14.000000 - 15.000000 = 26
 intervalle 15.000000 - 16.000000 = 10
 intervalle 16.000000 - 17.000000 = 25
 intervalle 17.000000 - 18.000000 = 15
 intervalle 18.000000 - 19.000000 = 21
 intervalle 19.000000 - 20.000000 = 14
```

```
Moyenne reelle = 10.314235
 intervalle 0.000000 - 1.000000 = 81
 intervalle 1.000000 - 2.000000 = 72
 intervalle 2.000000 - 3.000000 = 73
 intervalle 3.000000 - 4.000000 = 81
 intervalle 4.000000 - 5.000000 = 72
 intervalle 5.000000 - 6.000000 = 56
 intervalle 6.000000 - 7.000000 = 65
 intervalle 7.000000 - 8.000000 = 35
 intervalle 8.000000 - 9.000000 = 48
 intervalle 9.000000 - 10.000000 = 51
 intervalle 10.000000 - 11.000000 = 29
 intervalle 11.000000 - 12.000000 = 30
 intervalle 12.000000 - 13.000000 = 31
 intervalle 13.000000 - 14.000000 = 23
 intervalle 14.000000 - 15.000000 = 24
 intervalle 15.000000 - 16.000000 = 25
 intervalle 16.000000 - 17.000000 = 18
 intervalle 17.000000 - 18.000000 = 13
 intervalle 18.000000 - 19.000000 = 13
 intervalle 19.000000 - 20.000000 = 13
```

```
Moyenne reelle = 10.409414
 intervalle 0.000000 - 1.000000 = 109
 intervalle 1.000000 - 2.000000 = 80
 intervalle 2.000000 - 3.000000 = 82
 intervalle 3.000000 - 4.000000 = 61
 intervalle 4.000000 - 5.000000 = 59
 intervalle 5.000000 - 6.000000 = 72
 intervalle 6.000000 - 7.000000 = 50
 intervalle 7.000000 - 8.000000 = 47
 intervalle 8.000000 - 9.000000 = 41
 intervalle 9.000000 - 10.000000 = 31
 intervalle 10.000000 - 11.000000 = 31
 intervalle 11.000000 - 12.000000 = 33
 intervalle 12.000000 - 13.000000 = 29
 intervalle 13.000000 - 14.000000 = 20
 intervalle 14.000000 - 15.000000 = 17
 intervalle 15.000000 - 16.000000 = 17
 intervalle 16.000000 - 17.000000 = 17
 intervalle 17.000000 - 18.000000 = 12
 intervalle 18.000000 - 19.000000 = 19
 intervalle 19.000000 - 20.000000 = 11
```

```
Moyenne reelle = 9.834758
 intervalle 0.000000 - 1.000000 = 95
 intervalle 1.000000 - 2.000000 = 102
 intervalle 2.000000 - 3.000000 = 71
 intervalle 3.000000 - 4.000000 = 69
 intervalle 4.000000 - 5.000000 = 66
 intervalle 5.000000 - 6.000000 = 56
 intervalle 6.000000 - 7.000000 = 47
 intervalle 7.000000 - 8.000000 = 43
 intervalle 8.000000 - 9.000000 = 55
 intervalle 9.000000 - 10.000000 = 40
 intervalle 10.000000 - 11.000000 = 34
 intervalle 11.000000 - 12.000000 = 35
 intervalle 12.000000 - 13.000000 = 28
 intervalle 13.000000 - 14.000000 = 17
 intervalle 14.000000 - 15.000000 = 22
 intervalle 15.000000 - 16.000000 = 17
 intervalle 16.000000 - 17.000000 = 16
 intervalle 17.000000 - 18.000000 = 14
 intervalle 18.000000 - 19.000000 = 26
 intervalle 19.000000 - 20.000000 = 13
```

En conclusion, on obtient des distributions correctes cependant il aurait été de pouvoir comparer les résultats avec d'autres méthodes de simulations comme les méthodes de rejections.

Les distributions continues

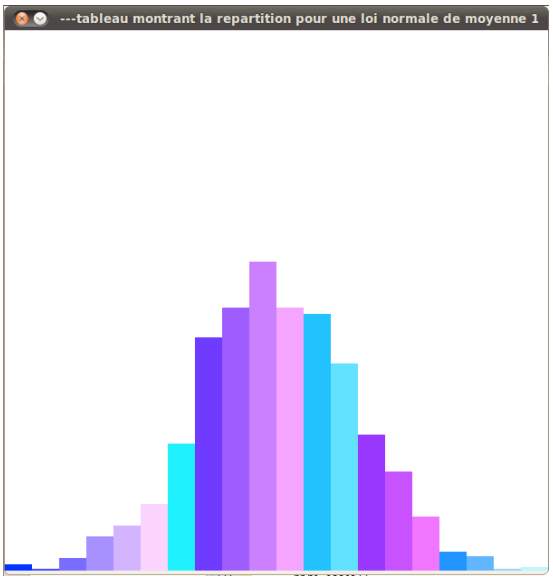
La loi normale avec la méthode de Box & Muller.

Contrairement aux deux lois précédentes, on ne peut pas connaître l'inverse de la loi normale Donc pour simuler cette distribution, il existe plusieurs méthodes dont celle de Box et Muller que l'on va implémenter dans ce tp.

code de la fonction :

```c
/**
 * \fn  int * loi_normale(float moyenne, float ecart_type)
 * \brief fonction qui effectue des tirages selon la loi normale utilise la methode de Box-Muller
 * \param moyenne moyenne de la loi normale
 * \param ecart_type ecart type de la loi normale
 * \param min valeur min tolere
 * \param max valeur maximale tolere
 * \return un tableau contenant le nombre de tirage effectue parmi des intervalles donn√©s
 *
 * Pour simuler ce fonctionnement, on utilise la methode de Box-Muller:
 * T1=sqrt(-2*ln(u1))*cos(2*pi*u2)
 * T1=sqrt(-2*ln(u1))*sin(2*pi*u2)
 * X1 = moyenne + ecart_type*T1
 * X2 = moyenne + ecart_type*T2
 */

int * loi_normale(double moyenne, double ecart_type, double min, double max)
{
   int i;
   int * tab;
   double x,y,u1,u2,moy=0.0,ecart=0.0;
   double tab_ecart[1000];

   tab = (int *) calloc((int)max,sizeof(int));
   if(!tab)
   {
      fprintf(stderr,"erreur alloc tab \n");
      exit(EXIT_FAILURE);
   }
   for(i=0;i<500;++i) /*500 car chaque iteration produit 2 tirages*/
   {
      /*generation des deux valeurs entre 0 et 1 */
      u1 = gene_rand();
      do
      {
         u2 = log(gene_rand());
      }while(isinf(u2));
      /*log(x) peut avoir un comportement indefini si u2 tend vers 0     */
      /*donc on teste la valeur tiree si elle est pas "egale" a l'infini */

      /*calcul des 2 tirages*/
      x = cos(2*PI*u1)*sqrt(-2.0*u2);
      y = sin(2*PI*u1)*sqrt(-2.0*u2);
      x = moyenne + sqrt(ecart_type)*x;
      y = moyenne + sqrt(ecart_type)*y;

      moy += x + y;
      tab_ecart[2*i] = x;
      tab_ecart[2*i+1] = y;

      if(x<max && x>min)
         tab[(int)x -(int)min]++;
      if(y<max && y>min)
         tab[(int)y -(int)min]++;
   }
   /*calcul de moyenne*/
   moy /= 1000,0;

   /*calcul de l'ecart type*/
   for(i=0;i<1000;++i)
      ecart += (tab_ecart[i] - moy)*(tab_ecart[i] - moy);
   ecart /= 1000.0;

   fprintf(stdout,"Moyenne reelle = %f \n",moy);
   fprintf(stdout,"Ecart-type reel = %f \n",ecart);
   return tab;
}
```

Relevé de mesure: les essais ont porté sur une loi normale de moyenne 10 et d'écart-type de 3
Chaque barre du graphe représente un intervalle de 1. (graphe représentant le 1er tirage)




```
Moyenne reelle = 9.949385
Ecart-type reel = 2.984993
 intervalle 0.000000 - 1.000000 = 1
 intervalle 1.000000 - 2.000000 = 1
 intervalle 2.000000 - 3.000000 = 10
 intervalle 3.000000 - 4.000000 = 17
 intervalle 4.000000 - 5.000000 = 23
 intervalle 5.000000 - 6.000000 = 37
 intervalle 6.000000 - 7.000000 = 70
 intervalle 7.000000 - 8.000000 = 99
 intervalle 8.000000 - 9.000000 = 116
 intervalle 9.000000 - 10.000000 = 111
 intervalle 10.000000 - 11.000000 = 144
 intervalle 11.000000 - 12.000000 = 129
 intervalle 12.000000 - 13.000000 = 91
 intervalle 13.000000 - 14.000000 = 66
 intervalle 14.000000 - 15.000000 = 45
 intervalle 15.000000 - 16.000000 = 20
 intervalle 16.000000 - 17.000000 = 7
 intervalle 17.000000 - 18.000000 = 8
 intervalle 18.000000 - 19.000000 = 3
 intervalle 19.000000 - 20.000000 = 1
```

```
Moyenne reelle = 9.992301
Ecart-type reel = 3.099496
 intervalle 0.000000 - 1.000000 = 3
 intervalle 1.000000 - 2.000000 = 4
 intervalle 2.000000 - 3.000000 = 6
 intervalle 3.000000 - 4.000000 = 9
 intervalle 4.000000 - 5.000000 = 28
 intervalle 5.000000 - 6.000000 = 44
 intervalle 6.000000 - 7.000000 = 59
 intervalle 7.000000 - 8.000000 = 101
 intervalle 8.000000 - 9.000000 = 138
 intervalle 9.000000 - 10.000000 = 122
 intervalle 10.000000 - 11.000000 = 124
 intervalle 11.000000 - 12.000000 = 105
 intervalle 12.000000 - 13.000000 = 85
 intervalle 13.000000 - 14.000000 = 66
 intervalle 14.000000 - 15.000000 = 49
 intervalle 15.000000 - 16.000000 = 27
 intervalle 16.000000 - 17.000000 = 17
 intervalle 17.000000 - 18.000000 = 8
 intervalle 18.000000 - 19.000000 = 4
 intervalle 19.000000 - 20.000000 = 0
```

```
Moyenne reelle = 10.097150
Ecart-type reel = 2.875797
 intervalle 0.000000 - 1.000000 = 0
 intervalle 1.000000 - 2.000000 = 3
 intervalle 2.000000 - 3.000000 = 4
 intervalle 3.000000 - 4.000000 = 13
 intervalle 4.000000 - 5.000000 = 18
 intervalle 5.000000 - 6.000000 = 45
 intervalle 6.000000 - 7.000000 = 58
 intervalle 7.000000 - 8.000000 = 96
 intervalle 8.000000 - 9.000000 = 107
 intervalle 9.000000 - 10.000000 = 136
 intervalle 10.000000 - 11.000000 = 147
 intervalle 11.000000 - 12.000000 = 112
 intervalle 12.000000 - 13.000000 = 107
 intervalle 13.000000 - 14.000000 = 73
 intervalle 14.000000 - 15.000000 = 43
 intervalle 15.000000 - 16.000000 = 18
 intervalle 16.000000 - 17.000000 = 12
 intervalle 17.000000 - 18.000000 = 4
 intervalle 18.000000 - 19.000000 = 3
 intervalle 19.000000 - 20.000000 = 0
```

```
Moyenne reelle = 9.926445
Ecart-type reel = 3.102281
 intervalle 0.000000 - 1.000000 = 1
 intervalle 1.000000 - 2.000000 = 3
 intervalle 2.000000 - 3.000000 = 11
 intervalle 3.000000 - 4.000000 = 13
 intervalle 4.000000 - 5.000000 = 22
 intervalle 5.000000 - 6.000000 = 58
 intervalle 6.000000 - 7.000000 = 68
 intervalle 7.000000 - 8.000000 = 99
 intervalle 8.000000 - 9.000000 = 117
 intervalle 9.000000 - 10.000000 = 126
 intervalle 10.000000 - 11.000000 = 124
 intervalle 11.000000 - 12.000000 = 100
 intervalle 12.000000 - 13.000000 = 91
 intervalle 13.000000 - 14.000000 = 61
 intervalle 14.000000 - 15.000000 = 58
 intervalle 15.000000 - 16.000000 = 26
 intervalle 16.000000 - 17.000000 = 13
 intervalle 17.000000 - 18.000000 = 7
 intervalle 18.000000 - 19.000000 = 0
 intervalle 19.000000 - 20.000000 = 2
```

```
Moyenne reelle = 9.984490
Ecart-type reel = 3.028029
 intervalle 0.000000 - 1.000000 = 0
 intervalle 1.000000 - 2.000000 = 3
 intervalle 2.000000 - 3.000000 = 10
 intervalle 3.000000 - 4.000000 = 17
 intervalle 4.000000 - 5.000000 = 30
 intervalle 5.000000 - 6.000000 = 38
 intervalle 6.000000 - 7.000000 = 64
 intervalle 7.000000 - 8.000000 = 77
 intervalle 8.000000 - 9.000000 = 120
 intervalle 9.000000 - 10.000000 = 136
 intervalle 10.000000 - 11.000000 = 141
 intervalle 11.000000 - 12.000000 = 99
 intervalle 12.000000 - 13.000000 = 113
 intervalle 13.000000 - 14.000000 = 65
 intervalle 14.000000 - 15.000000 = 43
 intervalle 15.000000 - 16.000000 = 23
 intervalle 16.000000 - 17.000000 = 10
 intervalle 17.000000 - 18.000000 = 6
 intervalle 18.000000 - 19.000000 = 4
 intervalle 19.000000 - 20.000000 = 1
```

En conclusion la méthode de Box & Muller donne une distribution correcte de la loi normale
On peut dire aussi, que cette méthode est efficace car elle nécessite seulement un tirage pour chaque nombre tirée contrairement aux autres méthodes du cours.

Les distributions continues

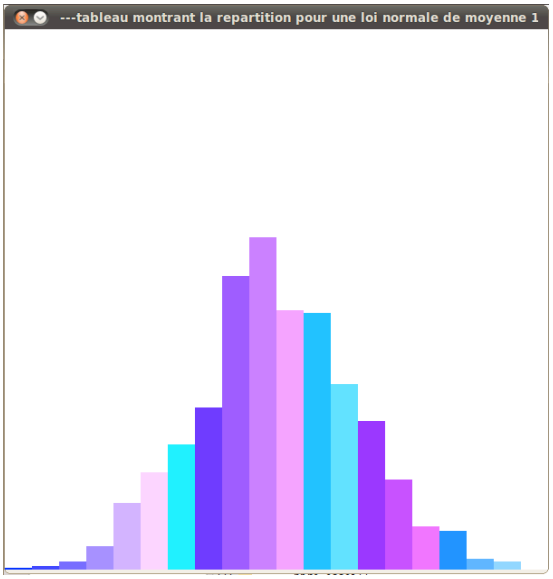
La loi normale avec la méthode dite de «réjection».

Contrairement aux deux lois précédentes, on ne peut pas connaître l'inverse de la loi normale Donc pour simuler cette distribution, il existe plusieurs méthodes dont celle dite de «réjection» que l'on va implémenter dans ce tp.

code de la fonction :

```c
/**
 * \fn  int * loi_normale_rejection(float moyenne, float ecart_type, float minX, float maxX)
 * \brief fonction qui effectue des tirages selon la loi normale utilise la methode de rejection
 * \param moyenne moyenne de la loi normale
 * \param ecart_type ecart type de la loi normale
 * \param min valeur min tolere
 * \param max valeur maximale tolere
 * \return un tableau contenant le nombre de tirage effectue parmi des intervalles donnÃ©s
 *
 * voir algo sujet
 */

int * loi_normale_rejection(double moyenne, double ecart_type, double minX, double maxX)
{
   double maxY,Na1,Na2,x,y,moy,ecart=0.0,tab_ecart[1000];
   int * tab,i,nbre_tir = 0, nbre_essai = 0;

   tab = (int *) calloc((int)maxX-(int)minX,sizeof(int));
   if(!tab)
   {
      fprintf(stderr,"erreur alloc tab \n");
      exit(EXIT_FAILURE);
   }

   /*ecart_type = sqrt(ecart_type);*/

   /*calcul de la norme sup de la distribution*/
   maxY = 1.0/(ecart_type*sqrt(2.0*PI));

   while(nbre_tir<1000)
   {
      Na1 = gene_rand();
      Na2 = gene_rand();

      x = minX + Na1 * (maxX - minX);
      y = maxY * Na2;

      if(maxY*exp(-0.5*((x-moyenne)/ecart_type)*((x-moyenne)/ecart_type)) > y)
      {
         tab[(int)x - (int)minX]++;
         tab_ecart[nbre_tir] = x;
         moy += x;
         nbre_tir++;
      }
      nbre_essai++;
   }

   /*calcul de moyenne*/
   moy /= 1000,0;

   /*calcul de l'ecart type*/
   for(i=0;i<1000;++i)
      ecart += (tab_ecart[i] - moy)*(tab_ecart[i] - moy);
   ecart /= 1000.0;
   ecart = sqrt(ecart);

   fprintf(stdout,"Moyenne reelle = %f \n",moy);
   fprintf(stdout,"Ecart-type reel = %f \n",ecart);
   return tab;
}
```

Relevé de mesure: les essais ont porté sur une loi normale de moyenne 10 et d'écart-type de 3
Chaque barre du graphe représente un intervalle de 1. (graphe représentant le 1er tirage)




```
Moyenne reelle = 9.785920
Ecart-type reel = 3.042343
 intervalle 0.000000 - 1.000000 = 1
 intervalle 1.000000 - 2.000000 = 3
 intervalle 2.000000 - 3.000000 = 9
 intervalle 3.000000 - 4.000000 = 9
 intervalle 4.000000 - 5.000000 = 33
 intervalle 5.000000 - 6.000000 = 53
 intervalle 6.000000 - 7.000000 = 73
 intervalle 7.000000 - 8.000000 = 94
 intervalle 8.000000 - 9.000000 = 112
 intervalle 9.000000 - 10.000000 = 137
 intervalle 10.000000 - 11.000000 = 136
 intervalle 11.000000 - 12.000000 = 129
 intervalle 12.000000 - 13.000000 = 67
 intervalle 13.000000 - 14.000000 = 57
 intervalle 14.000000 - 15.000000 = 43
 intervalle 15.000000 - 16.000000 = 22
 intervalle 16.000000 - 17.000000 = 9
 intervalle 17.000000 - 18.000000 = 7
 intervalle 18.000000 - 19.000000 = 3
 intervalle 19.000000 - 20.000000 = 3
```

```
Moyenne reelle = 9.956360
Ecart-type reel = 3.109554
 intervalle 0.000000 - 1.000000 = 1
 intervalle 1.000000 - 2.000000 = 3
 intervalle 2.000000 - 3.000000 = 11
 intervalle 3.000000 - 4.000000 = 13
 intervalle 4.000000 - 5.000000 = 27
 intervalle 5.000000 - 6.000000 = 43
 intervalle 6.000000 - 7.000000 = 75
 intervalle 7.000000 - 8.000000 = 94
 intervalle 8.000000 - 9.000000 = 117
 intervalle 9.000000 - 10.000000 = 103
 intervalle 10.000000 - 11.000000 = 135
 intervalle 11.000000 - 12.000000 = 115
 intervalle 12.000000 - 13.000000 = 99
 intervalle 13.000000 - 14.000000 = 70
 intervalle 14.000000 - 15.000000 = 42
 intervalle 15.000000 - 16.000000 = 28
 intervalle 16.000000 - 17.000000 = 13
 intervalle 17.000000 - 18.000000 = 7
 intervalle 18.000000 - 19.000000 = 2
 intervalle 19.000000 - 20.000000 = 2
```

```
Moyenne reelle = 10.056580
Ecart-type reel = 2.911651
 intervalle 0.000000 - 1.000000 = 2
 intervalle 1.000000 - 2.000000 = 1
 intervalle 2.000000 - 3.000000 = 3
 intervalle 3.000000 - 4.000000 = 11
 intervalle 4.000000 - 5.000000 = 18
 intervalle 5.000000 - 6.000000 = 39
 intervalle 6.000000 - 7.000000 = 59
 intervalle 7.000000 - 8.000000 = 106
 intervalle 8.000000 - 9.000000 = 116
 intervalle 9.000000 - 10.000000 = 141
 intervalle 10.000000 - 11.000000 = 145
 intervalle 11.000000 - 12.000000 = 120
 intervalle 12.000000 - 13.000000 = 89
 intervalle 13.000000 - 14.000000 = 62
 intervalle 14.000000 - 15.000000 = 38
 intervalle 15.000000 - 16.000000 = 26
 intervalle 16.000000 - 17.000000 = 11
 intervalle 17.000000 - 18.000000 = 4
 intervalle 18.000000 - 19.000000 = 8
 intervalle 19.000000 - 20.000000 = 1
```

```
Moyenne reelle = 10.138200
Ecart-type reel = 2.947255
 intervalle 0.000000 - 1.000000 = 0
 intervalle 1.000000 - 2.000000 = 2
 intervalle 2.000000 - 3.000000 = 5
 intervalle 3.000000 - 4.000000 = 17
 intervalle 4.000000 - 5.000000 = 18
 intervalle 5.000000 - 6.000000 = 41
 intervalle 6.000000 - 7.000000 = 53
 intervalle 7.000000 - 8.000000 = 89
 intervalle 8.000000 - 9.000000 = 120
 intervalle 9.000000 - 10.000000 = 138
 intervalle 10.000000 - 11.000000 = 132
 intervalle 11.000000 - 12.000000 = 125
 intervalle 12.000000 - 13.000000 = 96
 intervalle 13.000000 - 14.000000 = 75
 intervalle 14.000000 - 15.000000 = 30
 intervalle 15.000000 - 16.000000 = 34
 intervalle 16.000000 - 17.000000 = 15
 intervalle 17.000000 - 18.000000 = 7
 intervalle 18.000000 - 19.000000 = 2
 intervalle 19.000000 - 20.000000 = 1
```

```
Moyenne reelle = 10.054060
Ecart-type reel = 3.023779
 intervalle 0.000000 - 1.000000 = 1
 intervalle 1.000000 - 2.000000 = 3
 intervalle 2.000000 - 3.000000 = 5
 intervalle 3.000000 - 4.000000 = 14
 intervalle 4.000000 - 5.000000 = 23
 intervalle 5.000000 - 6.000000 = 42
 intervalle 6.000000 - 7.000000 = 75
 intervalle 7.000000 - 8.000000 = 93
 intervalle 8.000000 - 9.000000 = 110
 intervalle 9.000000 - 10.000000 = 115
 intervalle 10.000000 - 11.000000 = 130
 intervalle 11.000000 - 12.000000 = 113
 intervalle 12.000000 - 13.000000 = 110
 intervalle 13.000000 - 14.000000 = 76
 intervalle 14.000000 - 15.000000 = 42
 intervalle 15.000000 - 16.000000 = 22
 intervalle 16.000000 - 17.000000 = 17
 intervalle 17.000000 - 18.000000 = 6
 intervalle 18.000000 - 19.000000 = 2
 intervalle 19.000000 - 20.000000 = 1
```

En conclusion, on peut dire que cette méthode est moins efficace que celle de Box-Muller car moins proche des resultats théoriques et bien moins efficace due au nombre de tirage pour obtenir nos 1000 tirs.

Les library existantes

C/C++:

IMSL http://www.microsigma.fr/vnumerics/index.html

Java:

JMSL : http://www.vni.com/products/imsl/jmsl/ (library mathématiques générale)

JSL : http://cat.inist.fr/?aModele=afficheN&cpsidt=20854139

Conclusion

La génération est la base de la simulation, non seulement elle doit être fidèle à la théorie et de plus elle doit être efficace.

On remarque que cette génération peut être facile et juste pour des distributions connues, mai selle peut rapidement s'éloigner des résultats théoriques et devenir gourmande dès que la fonction de densité devient complexe et non-inversible.