TP Génération de nombres aléatoires

Télégraphique

Florian Meret, Etienne Guignard

Année 2016-2017

19/10/2016

Table des matières

Introduction

L'objectif de ce travail pratique est de générer des nombres pseudo aléatoires en utilisant différentes distributions (uniforme, exponentielle, Pareto) puis, en les affichant afin de pouvoir visualiser le nombre d'occurrences en fonction du tirage.

Calculs

Moyenne

On calcule la moyenne d'une variable numérique en additionnant les valeurs de toutes les observations incluses dans un ensemble de données, puis en divisant cette somme par le nombre d'observations qui font partie de l'ensemble. Ce calcul permet d'obtenir la valeur moyenne de toutes les données.

$$Moyenne = \frac{Somme\ de\ chaque\ élément}{Nombre\ d'éléments}$$

Variance

La variance est une mesure de la dispersion d'une série de données. Une variance faible indique que les nombres de la série de données sont proches l'un de l'autre. Au contraire, une variance élevée indique que les nombres sont très distants.

$$Variance = \frac{Somme\ de\ chaque\ élément^2}{Nombre\ d'élément} - moyenne^2$$

Écart type

L'écart-type est un paramètre de dispersion. Il mesure la dispersion de la série par rapport à la moyenne. Plus les éléments de la série sont éloignés de la moyenne, plus l'écart-type est élevé. Il est défini comme la racine carrée de la variance.

$$Ecart\ type = \sqrt{variance}$$

Distribution de Pareto

La distribution de Pareto a d'abord été proposée comme modèle pour la distribution des revenus. Elle est également utilisée comme modèle pour la répartition des populations d'une ville dans une zone donnée.

$$Distribution\ de\ Pareto = \frac{1}{Log(1 - rand + 10^{-12})} - 1$$

Distribution exponentielle

La distribution exponentielle est l'une des distributions continues largement employées. Elle est souvent utilisée pour modéliser le temps écoulé entre les événements.

$$Distribution\ exponentielle = -Log(1 - rand + 10^{-12})$$

Génération de nombres aléatoires

Pour générer plusieurs nombres aléatoires avec la formule ci-dessous, P1, P2, N et la première occurrence de x doivent au préalable être spécifiés afin que la génération soit possible. Pour que la séquence ne se répète pas, il est impératif que la valeur de x soit actualisée par le résultat du nombre aléatoire précèdent. Afin de trouver un nombre aléatoire que varie entre 0 et 1, nous avons donc divisé le nombre aléatoire par N.

$$Nombre\ aléatoire = P1 * x + P2\ mod\ N$$

Programmation

Afin de réaliser le travail demandé, nous avons décidé d'utiliser comme langage de programmation C#, car il implémente comme fonctionnalité de base la gestion de graphique ainsi qu'un générateur de nombres pseudo aléatoires facile à utiliser.

Voici le déroulement des différentes étapes du programme pour réaliser le travail demandé :

1. Initialisation
2. Boucle de tirage (10000 occurrences)
3. Génération des nombre aléatoires et utilisation des distributions
4. Calcule de la moyenne, l'écart type, la variance
5. Affichage du résultat et du graphique pour chaque distribution

Tirages

Pour chaque tirage, deux nombres aléatoires sont tirés. Le premier avec le générateur de nombres aléatoires de Windows et le second avec notre propre générateur de nombres aléatoires. A la suite de ça, le nombre aléatoire généré par notre fonction est utilisé par la distribution de Pareto et la distribution exponentielle. Finalement, chaque nombre généré est sommé puis mis au carré afin de calculer, à la fin de tous les tirages, la moyenne, la variance et l'écart type.

Histogrammes

Pour pouvoir comparer les différentes distributions, il est nécessaire de mémoriser tous les nombres générés. Nous avons décidé de procéder de la manière suivante : comme chaque valeur se trouve dans l'intervalle 0 et 1, on divise cet intervalle en 10 pour que l'on puisse représenter les valeurs sur un graphique. Pour ce faire, nous avons utilisé un tableau d'entier de taille 10 pour représenter tous les intervalles (0.0-0.1 ;0.1-0.2 ;0.2-0.3, etc…). L'ajout d'une valeur dans le tableau est simple. Si la valeur est entre l'intervalle 0.0-0.1, cela incrémente la valeur du premier indice du tableau correspondant à un intervalle 0.0-0.1. A la fin, quand toutes les valeurs sont ajoutées, elles sont affectées à la partie qui s'occupe de l'affichage.

Calcule du reste

Pour la distribution exponentielle ainsi que celle de Pareto, on remarque qu'il y a des tirages qui ne sont pas pris en compte dans l'intervalle 0 et 1. Par conséquent, nous avons décidé de récupérer les valeurs restantes pour les répartir proportionnellement sur chaque intervalle. Dans le calcul que nous effectuons, il reste au maximum 10 tirages non pris en compte. Cependant, ces 10 valeurs ne vont pas

changer significativement le résultat c'est pourquoi nous les avons laissés de côté. Voici la formule qui nous permet de calculer le reste pour chacun des intervalles :

$$intervalle[i] += Floor\left(\frac{intervalle[valeures\ restantes]}{nombre\ d'intervalles}\right)$$

Résultats

Distribution uniforme

Le générateur de nombres aléatoires de Windows n'est pas totalement aléatoire, car on remarque que toutes les barres correspondant à un intervalle de 0.1 ne sont pas uniformément réparties dans l'intervalle 0 et 1. Ce qui semble normal, car ce générateur n'est qu'un générateur de nombres pseudo aléatoires. Considérant le résultat de la variance de l'écart type et de la moyenne, on peut dire que notre résultat est bien conforme à ce que nous devrions retrouver pour ce type de générateur de nombres aléatoires.

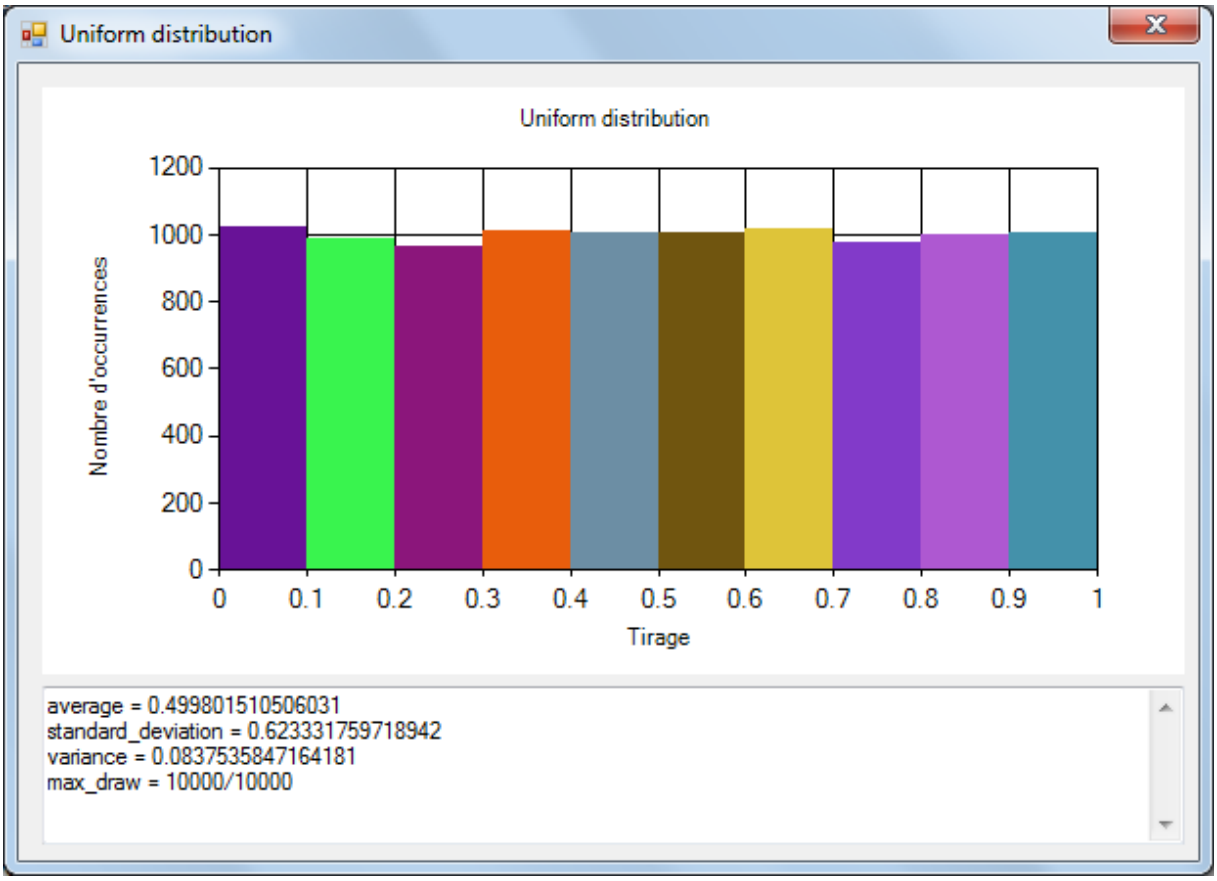

Figure 1: Graphique de distribution uniforme

Notre générateur de nombres aléatoires

Comme pour le générateur de nombres aléatoires de Windows, on constate que le nôtre n'est pas non plus parfaitement uniformément reparti dans l'intervalle 0 et 1. Nous générons par conséquent aussi des nombres pseudo aléatoires. Considérant le résultat de la variance de l'écart type et de la moyenne, on peut dire que notre résultat est bien conforme à ce que nous devrions retrouver pour ce type de générateur de nombres aléatoires.

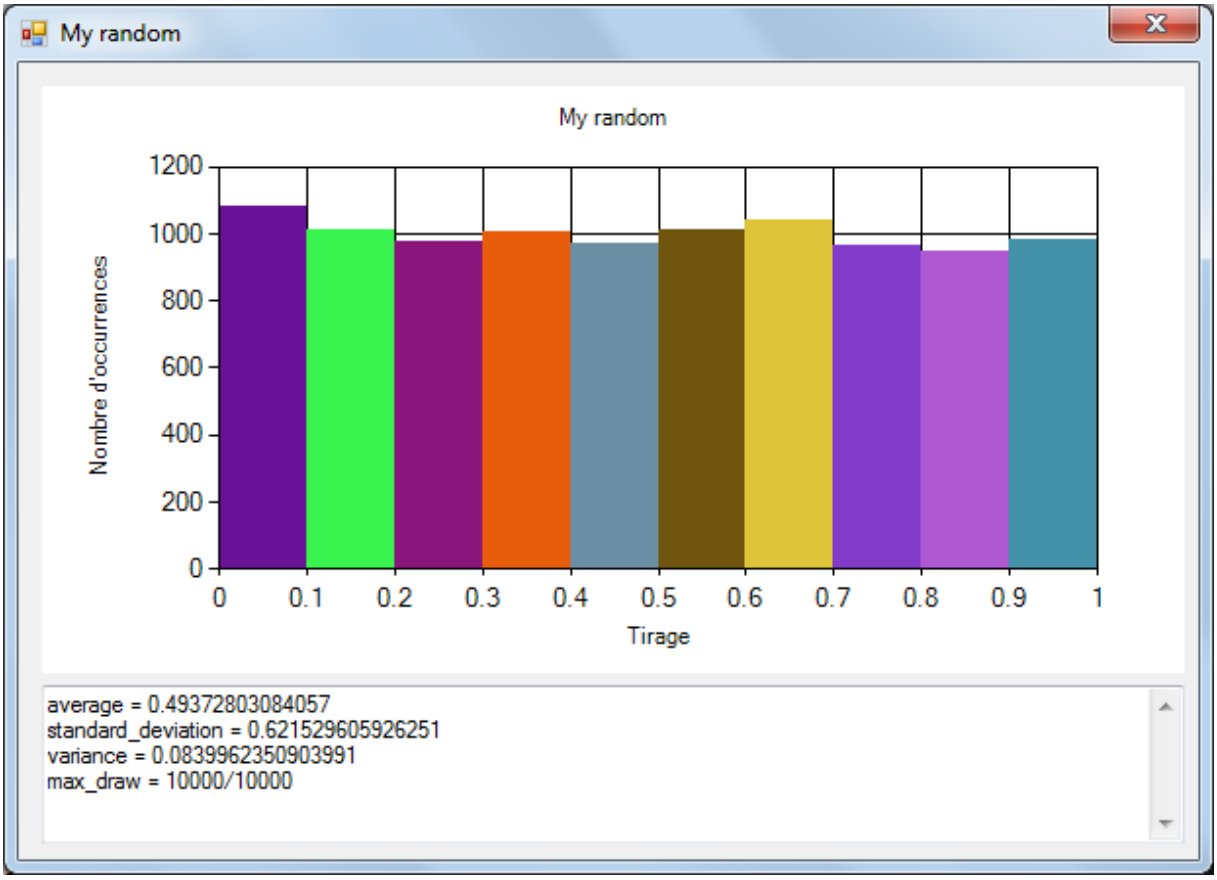

Figure 2: Graphique de notre générateur de nombres aléatoires

Distribution exponentielle

Contrairement à la distribution précédente, on sait que tous les tirages ne peuvent être inclus dans l'intervalle 0 et 1. Par conséquent, comme expliqué dans le point précèdent, nous avons ajouté le reste des tirages dans le graphique. Or, le graphique obtenu ci-dessous démontre bien une courbe exponentielle.

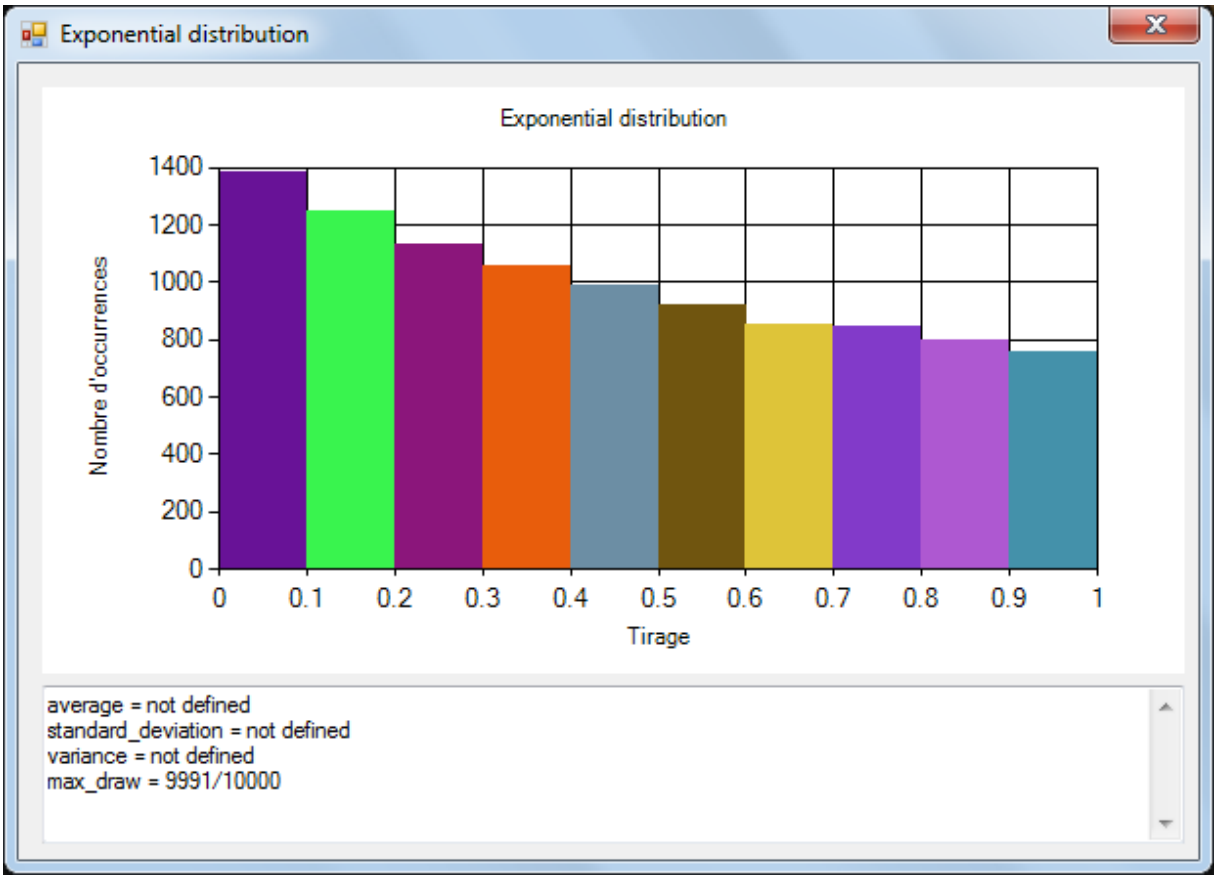

Figure 3: Graphique de distribution exponentielle

Distribution de Pareto

Contrairement à la distribution uniforme, on sait que tous les tirages ne peuvent être inclus dans l'intervalle 0 et 1. Par conséquent, comme expliqué dans le point précèdent, nous avons ajouté le reste des tirages dans le graphique.

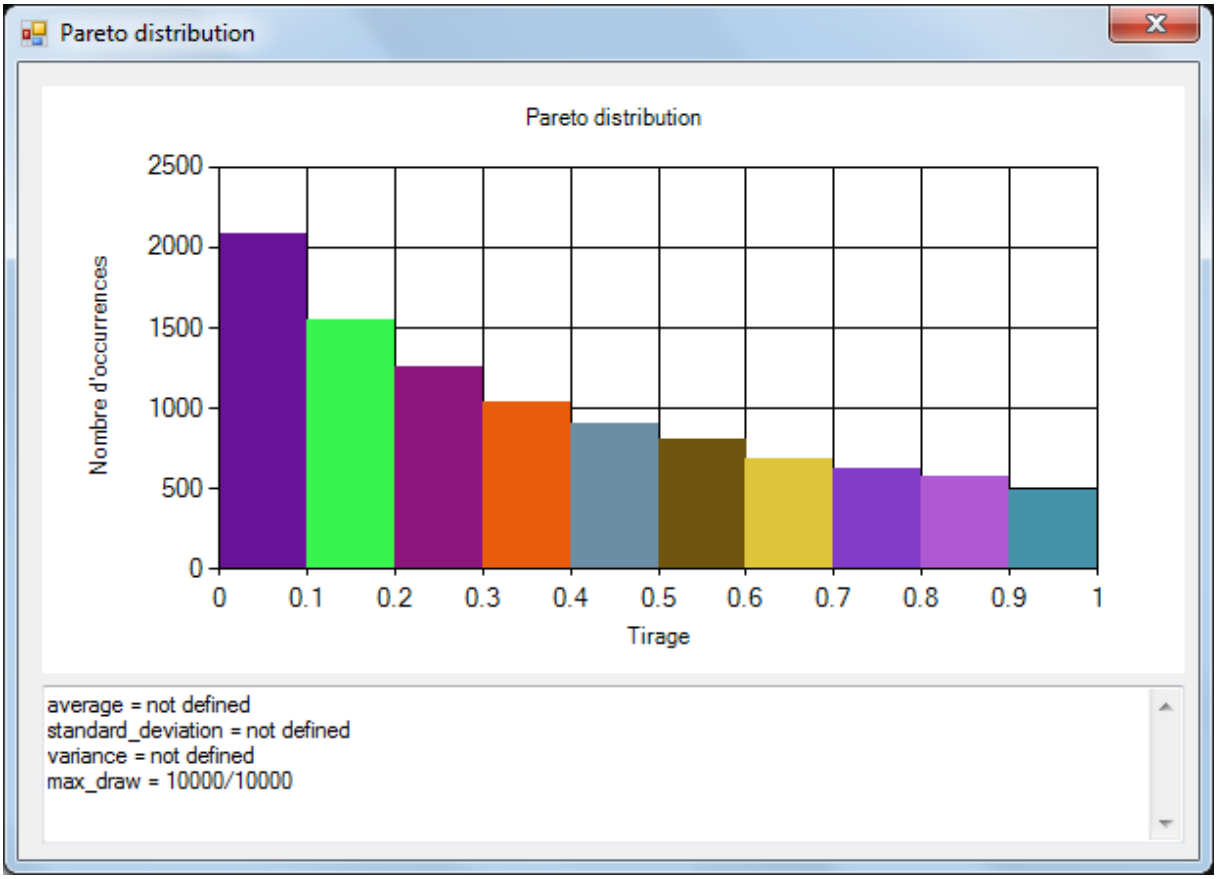

Figure 4: Graphique de distribution de Pareto

Conclusion

Dans ce travail, nous avons pu mettre en pratique la génération de nombres aléatoires. Ainsi, nous avons pris conscience des limites du phénomène de la génération de nombres aléatoires, qui n'est finalement pas totalement aléatoire. Nous avons aussi pu voir l'influence d'une valeur aléatoire sur différentes distributions telles que l'exponentielle et celle de Pareto.

Sources

- https://books.google.ch/books?id=L2JYZYI_l_wC&pg=PT575&lpg=PT575&dq=random+p1,+p2,+x0&source=bl&ots=9v0nKD8-E3&sig=YWarFLfmpBBXIXQASxOZvmzYv6w&hl=de&sa=X&ved=0ahUKEwjc5aem65bQAhULlxoKHU_cAfgQ6AEIKjAA#v=onepage&q=random%20p1%2C%20p2%2C%20x0&f=false
- http://www.pamvotis.org/vassis/RandGen.htm
- https://en.wikipedia.org/wiki/Exponential_distribution

```
using System;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace tp_random
{
    public partial class Result : Form
    {
        private string average = null;
        private string standard_deviation = null;
        private string variance = null;
        private string rand_type = null;
        private string max_draw = null;
        private int[] values = null;

        public string Average{
            get { return average; }
            set { average = value; }
        }
        public string Standard_deviation{
            get { return standard_deviation; }
            set { standard_deviation = value; }
        }
        public string Variance{
            get { return variance; }
            set { variance = value; }
        }
        public string Rand_type{
            get { return rand_type; }
            set { rand_type = value; }
        }
        public string Max_draw
        {
            get { return max_draw; }
            set { max_draw = value; }
        }
        public int[] Values{
            get { return values; }
            set { values = value; }
        }

        public Result() {
            InitializeComponent();
        }

        private void Result_Shown(object sender, EventArgs e) {

            // Form title
            this.Text = rand_type ?? "not defined";

            // Show if average, rand_type and variance
            TB_result.Text = text_result();

            // Display chart
            CH_values.Titles.Add(rand_type ?? "not defined");
            CH_values.ChartAreas[0].AxisX.Title = "Tirage";
            CH_values.ChartAreas[0].AxisY.Title = "Nombre d'occurrences";
            CH_values.Series[0].IsVisibleInLegend = false;
            double start = 0.05;
            CH_values.Series[0]["PixelPointWidth"] = "45";
            CH_values.ChartAreas[0].AxisX.Interval = 0.1;
            CH_values.ChartAreas[0].AxisX.Maximum = 1.0;
            CH_values.ChartAreas[0].AxisX.Minimum = 0.0;
            Random rnd = new Random();
            for (int i = 0; i < 10; i++) {
                CH_values.Series[0].Points.AddXY(start, values[i]);
                CH_values.Series[0].Points[i].Color = Color.FromArgb(rnd.Next(256),
                rnd.Next(256), rnd.Next(256)); ;
                start = start + 0.1 ;
            }
```

```
        }

        private string text_result() {
            StringBuilder sb = new StringBuilder();
            sb.AppendLine($"{nameof(average)} = {average ?? "not defined"}");
            sb.AppendLine($"{nameof(standard_deviation)} = {standard_deviation ?? "not
            defined"}");
            sb.AppendLine($"{nameof(variance)} = {variance ?? "not defined"}");
            sb.AppendLine($"{nameof(max_draw)} = {max_draw ?? "not defined"}");
            return sb.ToString();
        }
    }
}
```

```
using System;
using System.Windows.Forms;

/*
 * My random (pseudrandom number generator)
 * x[n+1] = (p1 * x[n] + p2) mod n
 * Sources:
 https://books.google.ch/books?id=L2JYZYI_l_wC&pg=PT575&lpg=PT575&dq=random+p1,+p2,+x0&source=
 bl&ots=9v0nKD8-E3&sig=YWarFLfmpBBXIXQASxOZvmzYv6w&hl=de&sa=X&ved=0ahUKEwjc5aem65bQAhULlxoKHU_
 cAfgQ6AEIKjAA#v=onepage&q=random%20p1%2C%20p2%2C%20x0&f=false
 *
 * Pareto and exponetial distribution
 * Sources: http://www.pamvotis.org/vassis/RandGen.htm
 */

namespace tp_random
{
    public partial class Form1 : Form
    {
        const double MAX_DRAW = 10000;

        public Form1() {
            InitializeComponent();
        }

        private void Form1_Shown(object sender, EventArgs e) {
            display_result();
        }

        private void display_result() {

            // Windows rand init
            double rand = 0;
            double sum = 0;
            double sum_squared = 0;
            Random random = new Random();

            // My rand init
            double my_rand = 0;
            double my_sum = 0;
            double my_sum_squared = 0;
            double seed = DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond;
            const double p1 = 16807, p2 = 0, n = uint.MaxValue;

            // Array init
            int[] uniform_array = new int[11];
            int[] my_rand_array = new int[11];
            int[] expo_array = new int[11];
            int[] pareto_array = new int[11];

            for (int i = 0; i < MAX_DRAW; i++) {

                // Windows random generation
                rand = uniform_distrib(random);
                sum += rand;
                sum_squared += Math.Pow(rand, 2);

                //My random generation
                my_rand = my_random(seed, n, p1, p2);
                seed = my_rand;
                my_rand = my_rand / n;
                my_sum += my_rand;
                my_sum_squared += Math.Pow(my_rand, 2);

                //Fill in array
                uniform_array = fill_in_array(uniform_array, rand);
                my_rand_array = fill_in_array(my_rand_array, my_rand);
                expo_array = fill_in_array(expo_array, expo_distrib(my_rand));
                pareto_array = fill_in_array(pareto_array, pareto_distrib(my_rand));
            }
```

```
            // Display result
            Result res_uniform_distrib = new Result();
            res_uniform_distrib.Rand_type = "Uniform distribution";
            res_uniform_distrib.Average = average(sum).ToString();
            res_uniform_distrib.Standard_deviation = standard_deviation(sum,
            sum_squared).ToString();
            res_uniform_distrib.Variance = variance(sum, sum_squared).ToString();
            res_uniform_distrib.Values = uniform_array;
            res_uniform_distrib.Max_draw = get_max_draw(uniform_array);
            res_uniform_distrib.Show();

            Result res_my_rand = new Result();
            res_my_rand.Rand_type = "My random";
            res_my_rand.Average = average(my_sum).ToString();
            res_my_rand.Standard_deviation = standard_deviation(my_sum,
            my_sum_squared).ToString();
            res_my_rand.Variance = variance(my_sum, my_sum_squared).ToString();
            res_my_rand.Values = my_rand_array;
            res_my_rand.Max_draw = get_max_draw(my_rand_array);
            res_my_rand.Show();

            expo_array = manage_reste_array(expo_array);
            Result res_expo_distrib = new Result();
            res_expo_distrib.Rand_type = "Exponential distribution";
            res_expo_distrib.Values = expo_array;
            res_expo_distrib.Max_draw = get_max_draw(expo_array);
            res_expo_distrib.Show();


            pareto_array = manage_reste_array(pareto_array);
            Result res_pareto_distrib = new Result();
            res_pareto_distrib.Rand_type = "Pareto distribution";
            res_pareto_distrib.Values = pareto_array;
            res_pareto_distrib.Max_draw = get_max_draw(pareto_array);
            res_pareto_distrib.Show();
        }

        private double my_random(double seed, double n, double p1, double p2) {
            return (p1 * seed + p2) % n;
        }

        private double uniform_distrib(Random rand) {
            return rand.NextDouble();
        }

        private double expo_distrib(double rand) {
            return -Math.Log((1 - rand + Math.Pow(10, -12)));
        }

        private double pareto_distrib(double rand) {
            return (1 / (Math.Sqrt(1 - rand + Math.Pow(10, -12)))) - 1;
        }

        private double average(double sum) {
            return sum / MAX_DRAW;
        }

        private double variance(double sum, double sum_squared) {
            return (sum_squared / MAX_DRAW) - Math.Pow(average(sum), 2);
        }

        private double standard_deviation(double sum, double sum_squared) {
            return Math.Sqrt(variance(sum_squared, sum));
        }

        // Compute the reste of the random value
        private int[] manage_reste_array(int[] array) {
            for (int i = 0; i < 10; i++) {
                array[i] += (int)Math.Floor((double)array[10] / 10);
            }
            return array;
```

```
        }

        // Add value in the array
        private int[] fill_in_array(int[] array, double value) {
            if (value > 0 && value < 0.1) {
                array[0]++;
            } else if (value > 0.1 && value < 0.2) {
                array[1]++;
            } else if (value > 0.2 && value < 0.3) {
                array[2]++;
            } else if (value > 0.3 && value < 0.4) {
                array[3]++;
            } else if (value > 0.4 && value < 0.5) {
                array[4]++;
            } else if (value > 0.5 && value < 0.6) {
                array[5]++;
            } else if (value > 0.6 && value < 0.7) {
                array[6]++;
            } else if (value > 0.7 && value < 0.8) {
                array[7]++;
            } else if (value > 0.8 && value < 0.9) {
                array[8]++;
            } else if (value > 0.9 && value <= 1) {
                array[9]++;
            }else {
                array[10]++;
            }
            return array;
        }

        // Get the number of used value on max draw
        private string get_max_draw(int[] array) {
            int total = 0;
            for (int i = 0; i < 10; i++) {
                total += array[i];
            }
            return $"{total.ToString()}/{MAX_DRAW}";
        }
    }
}
```