

C# Y .NET 8

Parte 9. Simulaciones

2024-12

Rafael Alberto Moreno Parra
ramsoftware@gmail.com

Contenido

Tabla de ilustraciones.....	3
Acerca del autor.....	4
Licencia de este libro	4
Licencia del software	4
Marcas registradas	5
En memoria de	6
Números aleatorios	7
Generadores de números aleatorios	9
Generador congruencial lineal.....	9
Blum Blum Shub	11
Pruebas a generadores de números aleatorios.....	12
Variables aleatorias	18
Distribuciones.....	19
Distribución Normal	19
Distribución Triangular	21
Distribución Uniforme.....	23
Problema del viajero.....	24
Resolviendo un sudoku	29
El problema de las tres puertas	33
Área bajo la curva	35
Dos robots se encuentran	42
Simulación de Inventarios	45
Simulación de Inventarios: Buscando el mejor valor	55
Jugadores y apuestas	63

Tabla de ilustraciones

Ilustración 1: Generando números aleatorios	8
Ilustración 2: Números aleatorios	10
Ilustración 3: Prueba del generador de números aleatorios	16
Ilustración 4: Prueba del generador de números aleatorios	17
Ilustración 5: Distribución Normal	19
Ilustración 6: Distribución Normal	20
Ilustración 7: Distribución Triangular	21
Ilustración 8: Distribución Triangular	22
Ilustración 9: Distribución Uniforme	23
Ilustración 10: Ruta más corta	28
Ilustración 11: Resuelve el Sudoku	32
Ilustración 12: Problema de las tres puertas	34
Ilustración 13: Curva $Y = F(X)$	35
Ilustración 14: Los límites X_{min} y X_{max} para hallar el área interna	36
Ilustración 15: Se dibuja un rectángulo que cubra la curva	37
Ilustración 16: Se lanzan puntos al azar al interior del rectángulo	38
Ilustración 17: Área hallada	41
Ilustración 18: Comparando con un servicio externo	41
Ilustración 19: Dos robots moviéndose	44
Ilustración 20: Archivo en Excel con las entradas del modelo de simulación	46
Ilustración 21: Semillas de los generadores de números pseudo-aleatorios	47
Ilustración 22: Comportamiento día a día de la simulación	48
Ilustración 23: Programa en C# que replica el comportamiento mostrado en Excel	48
Ilustración 24: Inicia la simulación	61
Ilustración 25: Finaliza la simulación	61
Ilustración 26: Valor en porcentaje a apostar	65

Acerca del autor

Rafael Alberto Moreno Parra

ramsoftware@gmail.com o enginelifelife@hotmail.com

Sitio Web: <http://darwin.50webs.com> (dedicado a la investigación de algoritmos evolutivos y vida artificial).

Github: <https://github.com/ramsoftware>

Youtube: <https://www.youtube.com/@RafaelMorenoP>

Licencia de este libro



Licencia del software

Todo el software desarrollado aquí tiene licencia LGPL "Lesser General Public License" [1]



Marcas registradas

En este libro se hace uso de las siguientes tecnologías registradas:

Microsoft ® Windows ® Enlace: <http://windows.microsoft.com/en-US/windows/home>

Microsoft ® Visual Studio 2022 ® Enlace: <https://visualstudio.microsoft.com/es/vs/>

En memoria de

En memoria de Tinita



Números aleatorios

.NET 8 tiene su propio generador de números pseudoaleatorios (no existe la aleatoriedad perfecta construida en un lenguaje de programación, para tal propósito, se requeriría hardware especializado que genere números aleatorios). El generador de .NET 8 requiere un valor semilla, por defecto, .NET 8 usa el reloj de la máquina. Cuando se trabaja con números aleatorios en modelos de simulación, entonces son números reales de tal manera que $0 \leq r < 1$, donde r es la denominación del número aleatorio

I/001.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Generando números al azar.

            //La semilla del generador la define el .NET
            Random azar = new();

            //Números enteros al azar
            Console.Write("Enteros positivos: ");
            for (int Contador = 1; Contador <= 10; Contador++) {
                int numEntero = azar.Next();
                Console.Write(numEntero + " | ");
            }

            //Números entre 0 y 1 al azar
            Console.Write("\r\n\r\nReales: ");
            for (int Contador = 1; Contador <= 10; Contador++) {
                double numReal = azar.NextDouble();
                Console.Write(numReal + " | ");
            }

            //Números entre 15 y 44 al azar.
            Console.Write("\r\n\r\nEnteros positivos entre 15 y 44: ");
            for (int Contador = 1; Contador <= 10; Contador++) {
                //El segundo parámetro debe ser +1 del
                //rango máximo que se busca
                int numEntero = azar.Next(15, 45);
                Console.Write(numEntero + " | ");
            }

            //Generando los mismos valores
            Console.Write("\r\n\r\nGenerando los mismos valores");
            Console.Write("al usar la misma semilla: ");
            Random AleatorioA = new(500);
            Random AleatorioB = new(500);
            for (int Contador = 1; Contador <= 20; Contador++) {
```

```

        int numA = AleatorioA.Next(55, 95);
        int numB = AleatorioB.Next(55, 95);
        Console.Write(numA + " y " + numB + " | ");
    }

    Console.WriteLine(" Final");
}
}
}

```

```

Enteros positivos: 2111923085 | 138467953 | 482820391 | 877967970 | 1550858139 | 2
034543776 | 784953033 | 24423575 | 969063847 | 1974149762 |

Reales: 0,7112869853854452 | 0,4797890592792097 | 0,8514211978628117 | 0,539018121
0406605 | 0,8079320879260405 | 0,6909123270727379 | 0,28510259183105124 | 0,664269
7127656781 | 0,6214570462263495 | 0,7830006306838707 |

Enteros positivos entre 15 y 44: 41 | 40 | 23 | 18 | 21 | 15 | 35 | 25 | 25 | 16 |

Generando los mismos valores al usar la misma semilla: 92 y 92 | 76 y 76 | 65 y 65
| 66 y 66 | 92 y 92 | 75 y 75 | 92 y 92 | 66 y 66 | 88 y 88 | 80 y 80 | 81 y 81 |
87 y 87 | 69 y 69 | 78 y 78 | 93 y 93 | 66 y 66 | 56 y 56 | 94 y 94 | 63 y 63 | 9
1 y 91 | Final

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\Ejemplo.exe (proceso

```

Ilustración 1: Generando números aleatorios

Generadores de números aleatorios

Es posible hacer un propio generador de números pseudoaleatorios, hay varios algoritmos que pueden ser útiles

(https://es.wikipedia.org/wiki/Generador_de_n%C3%BAmeros_pseudoaleatorios).

Generador congruencial lineal

Hace uso de la siguiente fórmula:

$$X_1 = (A * X_0 + B) \% N \quad r_1 = X_1 / N$$

$$X_2 = (A * X_1 + B) \% N \quad r_2 = X_2 / N$$

$$X_3 = (A * X_2 + B) \% N \quad r_3 = X_3 / N$$

$$X_n = (A * X_{n-1} + B) \% N$$

A, B, X_0 y N son valores llamados "semillas" que son ingresados por el usuario, a partir de ese punto el algoritmo genera los números pseudoaleatorios. Y la operación $\%$ retorna el residuo de la división.

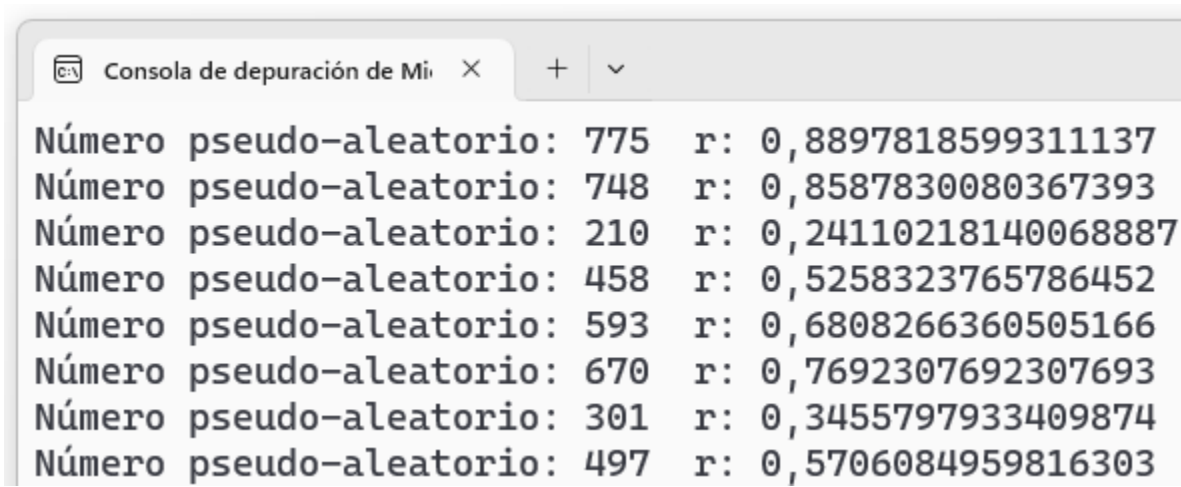
I/002.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Generador congruencial lineal
            long X0, A, B, N;

            //Valores de inicio. Se los da el usuario.
            X0 = 302;
            A = 278;
            B = 435;
            N = 871;

            for (int contador = 1; contador <= 100; contador++) {
                X0 = (A * X0 + B) % N;
                double r = (double) X0 / N;
                Console.WriteLine("Número pseudo-aleatorio: ");
                Console.WriteLine(X0 + "    r: " + r);
            }
        }
    }
}
```

```
}  
}  
}
```



The image shows a screenshot of a web browser's developer console. The console title bar reads 'Consola de depuración de Mi' followed by a close button (X) and expand/collapse icons (+ and v). The console contains eight lines of output, each showing a pseudo-random number and its corresponding decimal value. The text is as follows:

Número pseudo-aleatorio	r:
775	0,8897818599311137
748	0,8587830080367393
210	0,24110218140068887
458	0,5258323765786452
593	0,6808266360505166
670	0,7692307692307693
301	0,3455797933409874
497	0,5706084959816303

Ilustración 2: Números aleatorios

Blum Blum Shub

https://es.wikipedia.org/wiki/Blum_Blum_Shub

Hace uso de la siguiente fórmula:

$$X_1 = (X_0)^2 \% M \quad r_1 = X_1/M$$

$$X_2 = (X_1)^2 \% M \quad r_2 = X_2/M$$

$$X_3 = (X_2)^2 \% M \quad r_3 = X_3/M$$

$$X_n = (X_{n-1})^2 \% M$$

Donde M es el producto de la multiplicación de dos números primos muy grandes.

I/003.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Generador Blum Blum Shub
            long X0, M;

            //Valores de inicio. Se los da el usuario.
            X0 = 3;
            M = 11*19;

            for (int contador = 1; contador <= 100; contador++) {
                X0 = (X0*X0) % M;
                double r = (double) X0 / M;
                Console.Write("Número pseudo-aleatorio: ");
                Console.WriteLine(X0 + "   r: " + r);
            }
        }
    }
}
```

En el código se probó como demostración con números primos muy pequeños, eso se debe cambiar.

Pruebas a generadores de números aleatorios

¿Cómo saber si un generador de números aleatorios es aceptable? Hay diversas pruebas estadísticas para probar. A continuación, se muestran unas cuantas:

1. Los números generados deben dar como promedio un número muy cercano a 0.5
2. La varianza debe ser cercana a 1/12
3. Los números deben estar uniformemente distribuidos en el rango 0 a 1
4. Cada número es independiente del anterior y no afecta al siguiente

I/004.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main(string[] args) {

            List<double> Numeros = new List<double>();

            //Generador congruencial lineal
            long X0, A, B, Ndiv;

            //Valores de inicio. Se los da el usuario.
            X0 = 7302; //debe ser menor de Ndiv
            A = 5278;
            B = 3435;
            Ndiv = 20000;

            for (int contador = 1; contador <= 2000; contador++) {
                X0 = (A * X0 + B) % Ndiv;
                double r = (double)X0 / Ndiv;
                Numeros.Add(r);
            }

            //=====
            //Prueba de promedio
            //=====
            double Acumula = 0;
            for (int cont = 0; cont < Numeros.Count; cont++) {
                Acumula += Numeros[cont];
            }
            double Promedio = (double)Acumula / Numeros.Count;
            Console.Write("Promedio es: {0:0.00000}", Promedio);
            Console.WriteLine("y debe ser similar a 0,5");

            //=====
            //Prueba de varianza
            //=====
            double Sumatoria = 0;
```

```

for (int cont = 0; cont < Numeros.Count; cont++) {
    double Valor = (Numeros[cont] - Promedio);
    Sumatoria += Valor * Valor;
}
double Varianza = Sumatoria / Numeros.Count;
Console.Write("Varianza es: {0:0.00000} y debe", Varianza);
Console.WriteLine(" ser similar a {0:0.00000}", (1.0 / 12.0));

//=====
//Prueba de Uniformidad
//=====
int[] Rango = new int[10];
for (int cont = 0; cont < Numeros.Count; cont++) {
    if (Numeros[cont] < 0.1) Rango[0]++;
    else if (Numeros[cont] < 0.2) Rango[1]++;
    else if (Numeros[cont] < 0.3) Rango[2]++;
    else if (Numeros[cont] < 0.4) Rango[3]++;
    else if (Numeros[cont] < 0.5) Rango[4]++;
    else if (Numeros[cont] < 0.6) Rango[5]++;
    else if (Numeros[cont] < 0.7) Rango[6]++;
    else if (Numeros[cont] < 0.8) Rango[7]++;
    else if (Numeros[cont] < 0.9) Rango[8]++;
    else Rango[9]++;
}

Console.WriteLine("\n\nPrueba de Uniformidad");
Console.WriteLine("Números por rango");
Console.Write(" Rango\t");
Console.Write("\tFrecuencia Obtenida Fo");
Console.Write("\tFrecuencia Esperada Fe");
Console.WriteLine("\t((Fe-Fo)^2)/Fe");

double FEspera = Numeros.Count / 10;
double SumaRango = 0;
for (int cont = 0; cont < Rango.Length; cont++) {
    double Minimo = cont / 10.0;
    double Maximo = (cont + 1) / 10.0;
    double Valor = FEspera - Rango[cont];
    double Diferencia = (double)Valor * Valor / FEspera;
    Console.Write(" {0:0.0} y {1:0.0}", Minimo, Maximo);
    Console.Write("\t\t{0:#}\t\t\t{1:#}", Rango[cont], FEspera);
    Console.WriteLine("\t\t{0:0.00000}", Diferencia);

    SumaRango += Diferencia;
}

```

```

if (SumaRango < 16.9) {
    Console.Write("Pasa la prueba de uniformidad porque ");
    Console.WriteLine(SumaRango + " debe ser menor a 16.9");
}
else {
    Console.Write("NO pasó la prueba de uniformidad porque ");
    Console.WriteLine(SumaRango + " fue mayor o igual a 16.9");
}

//=====
//Prueba de Uniformidad de Kolmogorov-Smirnov
//=====
double[] ProbAcum = new double[10];
ProbAcum[0] = (double)Rango[0] / Numeros.Count;
for (int cont = 1; cont < Rango.Length; cont++) {
    double Valor = (double)Rango[cont] / Numeros.Count;
    ProbAcum[cont] = ProbAcum[cont - 1] + Valor;
}

Console.WriteLine("\n\nUniformidad de Kolmogorov-Smirnov");
Console.WriteLine("Probabilidad acumulada");
Console.WriteLine(" Rango\t\t\tEsperada\tObtenida\tDiferencia");
double MaxDiferencia = 0;
for (int cont = 0; cont < ProbAcum.Length; cont++) {
    double Minimo = cont / 10.0;
    double Maximo = (cont + 1) / 10.0;
    double Espera = (double)(cont + 1) / 10;
    double Diferencia = Math.Abs(ProbAcum[cont] - Espera);

    if (Diferencia > MaxDiferencia)
        MaxDiferencia = Diferencia;

    Console.Write(" {0:0.0} y {1:0.0}", Minimo, Maximo);
    Console.Write("\t\t{0:0.00}", Espera);
    Console.Write("\t\t{0:0.00}", ProbAcum[cont]);
    Console.WriteLine("\t\t{0:0.00}", Diferencia);
}

double Compara = 1.36 / Math.Sqrt(Numeros.Count);
if (MaxDiferencia < Compara) {
    Console.Write("Prueba de uniformidad aprobada porque ");
    Console.WriteLine(MaxDiferencia + " < " + Compara);
}
else {
    Console.WriteLine("NO aprobó prueba de uniformidad porque ");
    Console.WriteLine(MaxDiferencia + " >= " + Compara);
}

```

```

//=====
// Prueba de Independencia - Wald-Wolfowitz
//=====
Console.WriteLine("\n\nPrueba de Independencia - Wald-Wolfowitz");

//Deduce el valor de R, N1, N2, N
double N1 = 0, N2 = 0, R = 0;
int Bloque = 0;
for (int cont = 0; cont < Numeros.Count; cont++) {
    if (Numeros[cont] < Promedio) {
        if (Bloque != 1) R++;
        Bloque = 1;
        N1++;
    }
    else {
        if (Bloque != 2) R++;
        Bloque = 2;
        N2++;
    }
}

double N = Numeros.Count;

Console.WriteLine("N = " + N);
Console.WriteLine("N1 = " + N1);
Console.WriteLine("N2 = " + N2);
Console.WriteLine("R = " + R);

//Deduce la media
double Media = (2 * N1 * N2) / (N + 1);
double Variar = (Media - 1) * (Media - 2) / (N - 1);
double Z = (R - Media) / Math.Sqrt(Variar);

Console.WriteLine("Media = " + Media);
Console.WriteLine("Variación = " + Variar);
Console.WriteLine("Z = " + Z);

if (Z >= -1.96 && Z <= 1.96) {
    Console.Write("Pasa la prueba de independencia");
    Console.WriteLine("porque Z está entre -1.96 y 1.96");
}
else {
    Console.Write("NO pasó la prueba de independencia porque Z");
    Console.WriteLine(" está fuera del rango de -1.96 y 1.96");
}
}
}

```

}

Consola de depuración de Mi X + v

Promedio es: 0,50078y debe ser similar a 0,5
Varianza es: 0,08345 y debe ser similar a 0,08333

Prueba de Uniformidad

Números por rango

Rango	Frecuencia Obtenida Fo	Frecuencia Esperada Fe	$((Fe-Fo)^2)/Fe$
0,0 y 0,1	197	200	0,04500
0,1 y 0,2	206	200	0,18000
0,2 y 0,3	196	200	0,08000
0,3 y 0,4	203	200	0,04500
0,4 y 0,5	194	200	0,18000
0,5 y 0,6	204	200	0,08000
0,6 y 0,7	196	200	0,08000
0,7 y 0,8	204	200	0,08000
0,8 y 0,9	197	200	0,04500
0,9 y 1,0	203	200	0,04500

Pasa la prueba de uniformidad porque 0,86 debe ser menor a 16.9

Uniformidad de Kolmogorov-Smirnov

Probabilidad acumulada

Rango	Esperada	Obtenida	Diferencia
0,0 y 0,1	0,10	0,10	0,00
0,1 y 0,2	0,20	0,20	0,00
0,2 y 0,3	0,30	0,30	0,00
0,3 y 0,4	0,40	0,40	0,00
0,4 y 0,5	0,50	0,50	0,00
0,5 y 0,6	0,60	0,60	0,00
0,6 y 0,7	0,70	0,70	0,00
0,7 y 0,8	0,80	0,80	0,00
0,8 y 0,9	0,90	0,90	0,00
0,9 y 1,0	1,00	1,00	0,00

Prueba de uniformidad aprobada porque 0,00200000000000000018 < 0,03041052449399714

Ilustración 3: Prueba del generador de números aleatorios

Prueba de Independencia - Wald-Wolfowitz

N = 2000

N1 = 1000

N2 = 1000

R = 999

Media = 999,5002498750624

Variación = 498,25125000034365

Z = -0,022411080232072486

Pasa la prueba de independenciaporque Z está entre -1.96 y 1.96

Ilustración 4: Prueba del generador de números aleatorios

Variables aleatorias

Una variable aleatoria es el resultado de una ecuación que hace uso de números aleatorios, se representa así:

$$V = F(r)$$

Donde r es el número aleatorio entre 0 y 1. La variable aleatoria es usada dentro de los modelos de simulación, como la parte no controlable, pero si estimable. Por ejemplo, lanzar una moneda, puede generar cara o cruz. Ese valor es la variable aleatoria y puede expresarse de esta forma en C#:

```
Random Azar = new(); //Para el número aleatorio entre 0 y 1

//La variable aleatoria
if (Azar.NextDouble() < 0.5)
    return "Cara";
else
    return "Cruz";
```

Las funciones que pueden tener las variables aleatorias pueden ser discretas como la mostrada anteriormente con solo dos valores devueltos, que es "cara" o "cruz", o también pueden ser continuas, es decir, retornan un número real. A esas funciones continuas se les conocen como distribuciones. Ejemplo:

$$V = \textit{seno}(r)$$

Distribuciones

Distribución Normal

La distribución normal es una de las más utilizadas porque ciertos fenómenos tienen un comportamiento que sigue esta distribución:

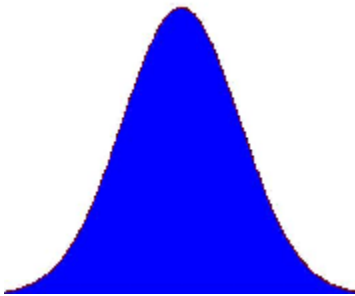


Ilustración 5: Distribución Normal

Los parámetros recibidos para poder usar esta distribución son M (media) y D (Desviación)

Para generar la variable aleatoria que utilice estos parámetros se usa la ecuación:

$$V = M + D * c$$

Donde c es un valor aleatorio que se deduce con la siguiente fórmula:

$$c = \cos(2\pi r_2) * \sqrt{-2 * \ln(r_1)}$$

Donde r_1 y r_2 son números aleatorios.

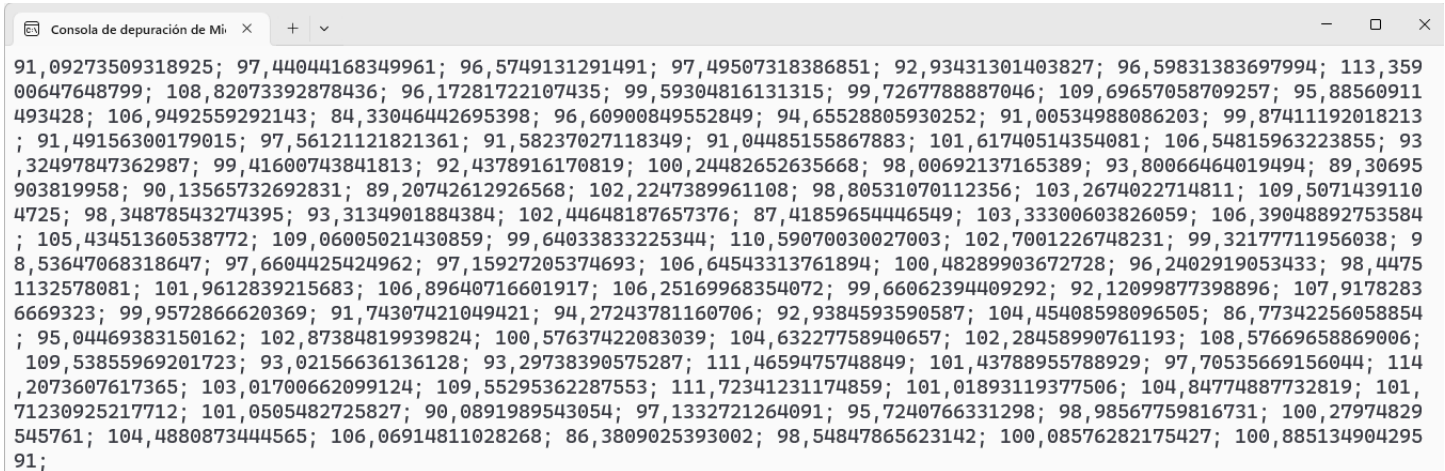
I/005.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main(string[] args) {
            /* Variable aleatoria
            Distribución Normal.
            Generar una variable aleatoria con media M y desviación D
            variable = M + D * c
            donde c = cos(2*PI*r2)*raizcuadrada(-2*LogarimoNatural(r1))
            r1 y r2 son números aleatorios */
            Random Azar = new();
            double M = 100;
            double D = 7;
            for (int cont = 1; cont <= 100; cont++) {
                double r1 = Azar.NextDouble();
```

```

double r2 = Azar.NextDouble();
double valA = Math.Cos(2 * Math.PI * r2);
double valB = Math.Sqrt(-2 * Math.Log(r1));
double c = valA * valB;
double variable = M + D * c;
Console.Write(variable + "; ");
    }
}
}
}

```



Consola de depuración de Mi x + v - □ ×

91,09273509318925; 97,44044168349961; 96,5749131291491; 97,49507318386851; 92,93431301403827; 96,59831383697994; 113,35900647648799; 108,82073392878436; 96,17281722107435; 99,59304816131315; 99,7267788887046; 109,69657058709257; 95,88560911493428; 106,9492559292143; 84,33046442695398; 96,60900849552849; 94,65528805930252; 91,00534988086203; 99,87411192018213; 91,49156300179015; 97,56121121821361; 91,58237027118349; 91,04485155867883; 101,61740514354081; 106,54815963223855; 93,32497847362987; 99,41600743841813; 92,4378916170819; 100,24482652635668; 98,00692137165389; 93,80066464019494; 89,30695903819958; 90,13565732692831; 89,20742612926568; 102,2247389961108; 98,80531070112356; 103,2674022714811; 109,50714391104725; 98,34878543274395; 93,3134901884384; 102,44648187657376; 87,41859654446549; 103,33300603826059; 106,39048892753584; 105,43451360538772; 109,06005021430859; 99,64033833225344; 110,59070030027003; 102,7001226748231; 99,32177711956038; 98,53647068318647; 97,6604425424962; 97,15927205374693; 106,64543313761894; 100,48289903672728; 96,2402919053433; 98,44751132578081; 101,9612839215683; 106,89640716601917; 106,25169968354072; 99,66062394409292; 92,12099877398896; 107,91782836669323; 99,9572866620369; 91,74307421049421; 94,27243781160706; 92,9384593590587; 104,45408598096505; 86,77342256058854; 95,04469383150162; 102,87384819939824; 100,57637422083039; 104,63227758940657; 102,28458990761193; 108,57669658869006; 109,53855969201723; 93,02156636136128; 93,29738390575287; 111,4659475748849; 101,43788955788929; 97,70535669156044; 114,2073607617365; 103,01700662099124; 109,55295362287553; 111,72341231174859; 101,01893119377506; 104,84774887732819; 101,71230925217712; 101,0505482725827; 90,0891989543054; 97,1332721264091; 95,7240766331298; 98,98567759816731; 100,27974829545761; 104,4880873444565; 106,06914811028268; 86,3809025393002; 98,54847865623142; 100,08576282175427; 100,88513490429591;

Ilustración 6: Distribución Normal

Distribución Triangular



Ilustración 7: Distribución Triangular

Los parámetros recibidos para poder usar esta distribución son un valor mínimo (a), un valor más probable (b) y un valor máximo (c)

Para calcular el valor de la variable aleatorio está el siguiente algoritmo:

Si $r < \frac{b-a}{c-a}$ entonces

$$V = a + \sqrt{r * (c - a) * (b - a)}$$

De lo contrario

$$V = c - \sqrt{(1 - r) * (c - a) * (c - b)}$$

Donde r es un número aleatorio.

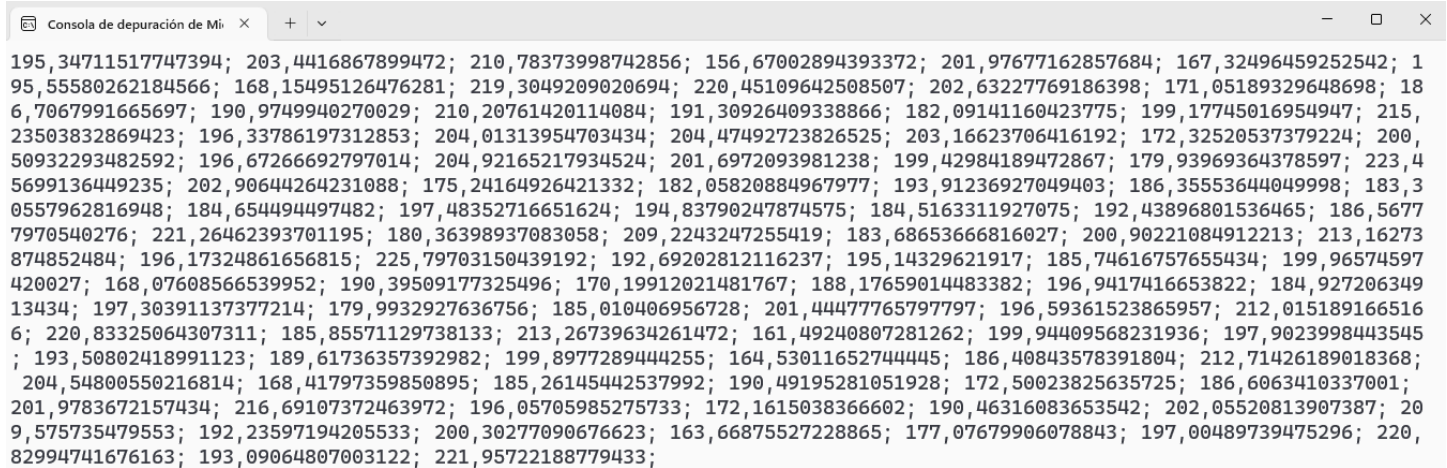
I/006.cs

```
namespace Ejemplo {  
    internal class Program {  
        static void Main(string[] args) {  
            /* Distribución Triangular. Generar una variable aleatoria  
            con valor mínimo A,  
            valor más probable B y valor máximo C  
            Si r < (B-A)/(C-A)  
                Variable = A + raizcuadrada(r*(C-A)*(B-A))  
            de lo contrario  
                Variable = C - raizcuadrada((1-r)*(C-A)*(C-B))  
            */  
            Random Azar = new();  
  
            double A = 150;
```

```

double B = 190;
double C = 230;
double variable = 0;
for (int cont = 1; cont <= 100; cont++) {
    double r = Azar.NextDouble();
    if (r < (B - A) / (C - A))
        variable = A + Math.Sqrt(r * (C - A) * (B - A));
    else
        variable = C - Math.Sqrt((1 - r) * (C - A) * (C - B));
    Console.Write(variable + "; ");
}
}
}
}

```



Consola de depuración de Mi...

195,34711517747394; 203,4416867899472; 210,78373998742856; 156,67002894393372; 201,97677162857684; 167,32496459252542; 195,55580262184566; 168,15495126476281; 219,3049209020694; 220,45109642508507; 202,63227769186398; 171,05189329648698; 186,7067991665697; 190,9749940270029; 210,20761420114084; 191,30926409338866; 182,09141160423775; 199,17745016954947; 215,23503832869423; 196,33786197312853; 204,01313954703434; 204,47492723826525; 203,16623706416192; 172,32520537379224; 200,50932293482592; 196,67266692797014; 204,92165217934524; 201,6972093981238; 199,42984189472867; 179,93969364378597; 223,45699136449235; 202,90644264231088; 175,24164926421332; 182,05820884967977; 193,91236927049403; 186,35553644049998; 183,30557962816948; 184,654494497482; 197,48352716651624; 194,83790247874575; 184,5163311927075; 192,43896801536465; 186,56777970540276; 221,26462393701195; 180,36398937083058; 209,2243247255419; 183,68653666816027; 200,90221084912213; 213,16273874852484; 196,17324861656815; 225,79703150439192; 192,69202812116237; 195,14329621917; 185,74616757655434; 199,96574597420027; 168,07608566539952; 190,39509177325496; 170,19912021481767; 188,17659014483382; 196,9417416653822; 184,92720634913434; 197,30391137377214; 179,9932927636756; 185,010406956728; 201,44477765797797; 196,59361523865957; 212,0151891665166; 220,83325064307311; 185,85571129738133; 213,26739634261472; 161,49240807281262; 199,94409568231936; 197,9023998443545; 193,50802418991123; 189,61736357392982; 199,8977289444255; 164,53011652744445; 186,40843578391804; 212,71426189018368; 204,54800550216814; 168,41797359850895; 185,26145442537992; 190,49195281051928; 172,50023825635725; 186,6063410337001; 201,9783672157434; 216,69107372463972; 196,05705985275733; 172,1615038366602; 190,46316083653542; 202,05520813907387; 209,575735479553; 192,23597194205533; 200,30277090676623; 163,66875527228865; 177,07679906078843; 197,00489739475296; 220,82994741676163; 193,09064807003122; 221,95722188779433;

Ilustración 8: Distribución Triangular

Distribución Uniforme

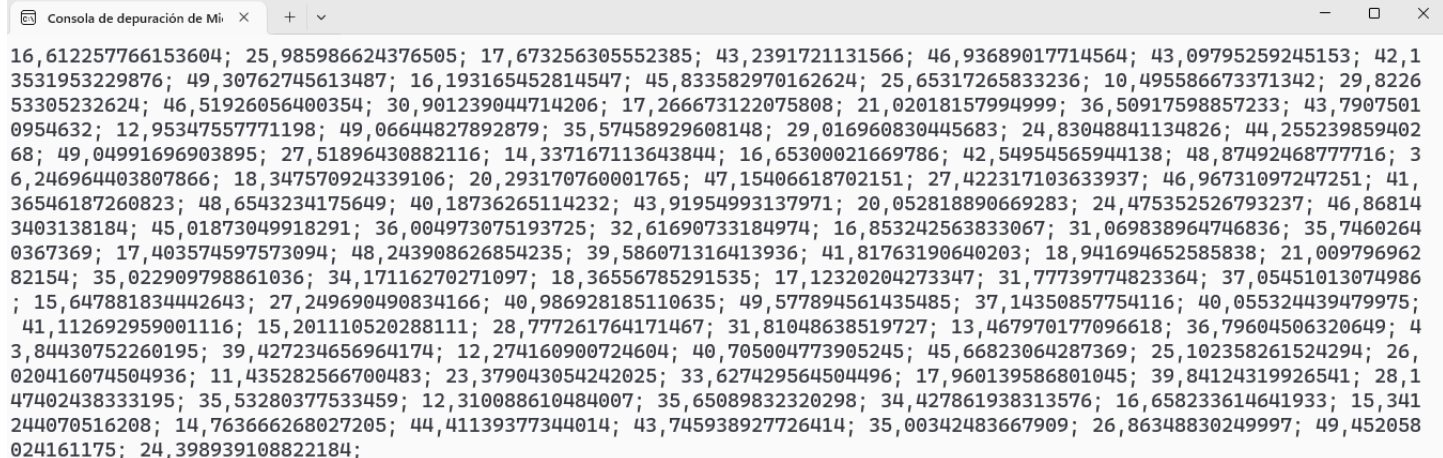
Se requiere un valor aleatorio entre A y B, y que su ocurrencia sea uniforme entre esos dos valores.

$$V = r * (B - A) + A$$

Donde r es un número aleatorio.

I/007.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main(string[] args) {
            /* Variable aleatoria
            Distribución uniforme. Generar un número
            entero entre A y B
            variable = r * (B - A) + A */
            Random Azar = new Random();
            double A = 10;
            double B = 50;
            for (int cont = 1; cont <= 100; cont++) {
                double r = Azar.NextDouble();
                double variable = r * (B - A) + A;
                Console.Write(variable + "; ");
            }
        }
    }
}
```



```
16,612257766153604; 25,985986624376505; 17,673256305552385; 43,2391721131566; 46,93689017714564; 43,09795259245153; 42,1
3531953229876; 49,30762745613487; 16,193165452814547; 45,833582970162624; 25,65317265833236; 10,495586673371342; 29,8226
53305232624; 46,51926056400354; 30,901239044714206; 17,266673122075808; 21,02018157994999; 36,50917598857233; 43,7907501
0954632; 12,95347557771198; 49,06644827892879; 35,57458929608148; 29,016960830445683; 24,83048841134826; 44,255239859402
68; 49,04991696903895; 27,51896430882116; 14,337167113643844; 16,65300021669786; 42,54954565944138; 48,87492468777716; 3
6,246964403807866; 18,347570924339106; 20,293170760001765; 47,15406618702151; 27,422317103633937; 46,96731097247251; 41,
36546187260823; 48,6543234175649; 40,18736265114232; 43,91954993137971; 20,052818890669283; 24,475352526793237; 46,86814
3403138184; 45,01873049918291; 36,004973075193725; 32,61690733184974; 16,853242563833067; 31,069838964746836; 35,7460264
0367369; 17,403574597573094; 48,243908626854235; 39,586071316413936; 41,81763190640203; 18,941694652585838; 21,009796962
82154; 35,022909798861036; 34,17116270271097; 18,36556785291535; 17,12320204273347; 31,77739774823364; 37,05451013074986
; 15,647881834442643; 27,249690490834166; 40,986928185110635; 49,577894561435485; 37,14350857754116; 40,055324439479975;
41,112692959001116; 15,201110520288111; 28,777261764171467; 31,81048638519727; 13,467970177096618; 36,79604506320649; 4
3,84430752260195; 39,427234656964174; 12,274160900724604; 40,705004773905245; 45,66823064287369; 25,102358261524294; 26,
020416074504936; 11,435282566700483; 23,379043054242025; 33,627429564504496; 17,960139586801045; 39,84124319926541; 28,1
47402438333195; 35,53280377533459; 12,310088610484007; 35,65089832320298; 34,427861938313576; 16,658233614641933; 15,341
244070516208; 14,763666268027205; 44,41139377344014; 43,745938927726414; 35,00342483667909; 26,86348830249997; 49,452058
024161175; 24,398939108822184;
```

Ilustración 9: Distribución Uniforme

Problema del viajero

Hay varias ciudades por visitar, sólo puede visitarse una por vez y cada viaje de una ciudad a otra tiene un costo, inclusive ir de la ciudad C a D puede tener un valor distinto de ir de D a C. El viajero debe planificar el viaje de tal manera que visite todas las ciudades, sólo una vez por ciudad y que le salga lo más económico posible.

A continuación, una tabla de costos de viaje de ir de una ciudad (vertical) a otra ciudad (horizontal). Por ejemplo, ir de F a B cuesta \$37, o ir de K a E cuesta \$24

Ciudad	A	B	C	D	E	F	G	H	I	J	K
A	\$ 0	\$ 74	\$ 13	\$ 10	\$ 75	\$ 49	\$ 21	\$ 60	\$ 60	\$ 97	\$ 89
B	\$ 43	\$ 0	\$ 31	\$ 86	\$ 41	\$ 92	\$ 75	\$ 56	\$ 72	\$ 41	\$ 10
C	\$ 67	\$ 39	\$ 0	\$ 64	\$ 88	\$ 75	\$ 82	\$ 50	\$ 68	\$ 14	\$ 38
D	\$ 12	\$ 52	\$ 28	\$ 0	\$ 16	\$ 32	\$ 51	\$ 77	\$ 85	\$ 67	\$ 28
E	\$ 22	\$ 85	\$ 19	\$ 51	\$ 0	\$ 51	\$ 19	\$ 11	\$ 37	\$ 62	\$ 63
F	\$ 10	\$ 37	\$ 13	\$ 35	\$ 12	\$ 0	\$ 43	\$ 53	\$ 49	\$ 98	\$ 77
G	\$ 18	\$ 75	\$ 62	\$ 64	\$ 32	\$ 58	\$ 0	\$ 81	\$ 86	\$ 93	\$ 46
H	\$ 22	\$ 37	\$ 51	\$ 41	\$ 12	\$ 71	\$ 76	\$ 0	\$ 92	\$ 99	\$ 15
I	\$ 36	\$ 33	\$ 71	\$ 25	\$ 92	\$ 10	\$ 93	\$ 52	\$ 0	\$ 98	\$ 61
J	\$ 38	\$ 94	\$ 39	\$ 52	\$ 34	\$ 10	\$ 58	\$ 92	\$ 87	\$ 0	\$ 26
K	\$ 59	\$ 32	\$ 30	\$ 92	\$ 24	\$ 12	\$ 51	\$ 36	\$ 38	\$ 98	\$ 0

Una idea de un trayecto sería este: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G \rightarrow H \rightarrow I \rightarrow J \rightarrow K$

El costo del viaje sería: $\$74 + \$31 + \$64 + \$16 + \$51 + \$43 + \$81 + \$92 + \$98 + \$26 = \$576$

Si se modificara el viaje a: $B \rightarrow A \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G \rightarrow H \rightarrow I \rightarrow J \rightarrow K$

El costo del viaje sería: $\$43 + \$13 + \$64 + \$16 + \$51 + \$43 + \$81 + \$92 + \$98 + \$26 = \$527$

Una ruta más económica.

¿Cuántas rutas posibles habría? Es la función factorial del total de ciudades, en el ejemplo serían $11!$ Rutas = 39.916.800 rutas. Y de todas esas rutas habrá una que será la más económica, pero hay que hacer cálculo de costos por cada ruta. Si fuesen unas 20 ciudades, las rutas posibles serían: 2.432.902.008.176.640.000, haciendo cálculos, y teniendo un computador que sea capaz de calcular el costo de 1000 millones de rutas por segundo, tardaría 2.432.902.008 segundos aproximadamente en evaluar todas las rutas, que equivalen a aproximadamente 77 años. Eso es muchísimo tiempo.

El método propuesto es mucho más rápido y da con una ruta con un costo bajo (no el más bajo, pero si lo suficiente). Tarda pocos segundos. ¿En qué consiste? Parte de una ruta inicial a la cual se le calcula su costo, al azar se toman dos ciudades de la ruta y se intercambian, se vuelve a calcular el nuevo costo, si es menor, entonces se deja la nueva ruta, caso contrario, se repone como estaba anteriormente. Se repite ese proceso por el tiempo deseado, y la última ruta hallada será lo mejor encontrado por el momento.


```

namespace Ejemplo {
    internal class Program {
        static void Main(string[] args) {
            /* Hay N ciudades a recorrer (0 a N-1).
               Sólo se puede visitar una ciudad por vez.
               En la tabla aparece cuanto cuesta ir de una
               ciudad (origen) a otra ciudad (destino).
               ¿Qué ruta tomar para visitar todas las
               ciudades con el mínimo costo? */
            Random Azar = new();

            int ciudad = 20; //Número de ciudades

            //Valor mínimo que tendrá ir de una ciudad a otra
            int minValor = 15;

            //Valor máximo que tendrá ir de una ciudad a otra
            int maxValor = 85;

            //Ciudad origen a Ciudad destino
            int dest1, dest2;

            //Genera valores de viaje al azar
            int[,] valorviajes = Valores(ciudad, minValor, maxValor);

            //Imprime los valores
            Imprime(valorviajes);

            //Inicia con una ruta predeterminada 0, 1, 2, 3, .... N
            int[] ruta = IniciaRuta(ciudad);

            //Deduce el costo de esa ruta predeterminada
            int costo = DeduceCosto(ruta, valorviajes);

            Console.WriteLine("\r\nRutas:");
            ImprimeRuta(ruta, costo);

            //Usando el método Monte Carlo se buscarán
            //otras rutas con menor costo
            for (int pruebas = 1; pruebas <= 700000; pruebas++) {
                double r1 = Azar.NextDouble();
                dest1 = (int)Math.Floor( r1 * ruta.Length);

                do {
                    double r2 = Azar.NextDouble();
                    dest2 = (int)Math.Floor(r2 * ruta.Length);
                } while (dest2 == dest1);
            }
        }
    }
}

```

```

ModificaRuta(ruta, dest1, dest2);

int costoNuevo = DeduceCosto(ruta, valorviajes);

if (costoNuevo < costo) {
    costo = costoNuevo;
    ImprimeRuta(ruta, costo);
}
else {
    //Dejar la ruta como antes
    ModificaRuta(ruta, dest1, dest2);
}
}
}

//Modifica la ruta de viaje
static void ModificaRuta(int[] ruta, int dest1, int dest2) {
    int tmp = ruta[dest1];
    ruta[dest1] = ruta[dest2];
    ruta[dest2] = tmp;
}

//Inicia el arreglo bidimensional de rutas
static int[] IniciaRuta(int limite) {
    int[] ruta = new int[limite];
    for (int cont = 0; cont < limite; cont++)
        ruta[cont] = cont;
    return ruta;
}

//Deduce el costo de la ruta de viaje
static int DeduceCosto(int[] ruta, int[,] costos) {
    int acum = 0;
    for (int cont = 0; cont < ruta.Length - 1; cont++)
        acum += costos[ruta[cont], ruta[cont + 1]];
    return acum;
}

//Llena de valores al azar la tabla de costos de viajes
//de una ciudad a otra
static int[,] Valores(int ciudad, int minValor, int maxValor) {
    int[,] tablero = new int[ciudad, ciudad];
    Random Azar = new();
    for (int fila = 0; fila < ciudad; fila++) {
        for (int columna = 0; columna < ciudad; columna++) {
            tablero[fila, columna] = Azar.Next(minValor, maxValor);
        }
    }
}

```

```

        tablero[fila, fila] = 0;
    }
    return tablero;
}

//Imprime el tablero
static void Imprime(int[,] tablero) {
    Console.WriteLine("Tabla de costos");
    Console.Write("    ");
    for (int col = 0; col < tablero.GetLength(1); col++)
        Console.Write((char)(col + 65) + " ");
    Console.WriteLine(" ");

    for (int fila = 0; fila < tablero.GetLength(0); fila++) {
        Console.Write((char)(fila + 65) + ": ");
        for (int col = 0; col < tablero.GetLength(1); col++)
            Console.Write(tablero[fila, col] + " ");
        Console.WriteLine(" ");
    }
}

//Imprime la ruta y su costo
static void ImprimeRuta(int[] ruta, int costo) {
    for (int col = 0; col < ruta.Length; col++) {
        Console.Write((char)(ruta[col] + 65) + "->");
    }
    Console.WriteLine(" $" + costo);
}
}
}

```

La ejecución del programa genera una tabla de costos al azar para unas 20 ciudades, luego muestra el costo de una ruta inicial y el proceso para encontrar rutas de menor costo cada vez.

Tabla de costos

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
A:	0	56	80	26	64	47	51	27	38	58	58	42	15	20	62	54	62	58	75	34
B:	36	0	20	58	59	47	18	65	23	33	36	46	25	33	58	32	45	41	21	44
C:	71	48	0	32	20	50	55	83	56	21	62	48	48	43	56	47	33	77	34	19
D:	63	37	76	0	16	21	65	60	44	17	81	24	43	71	43	53	64	28	49	66
E:	16	50	64	71	0	79	57	82	78	84	20	84	66	39	76	61	49	77	28	58
F:	60	61	65	61	24	0	27	57	68	25	61	48	81	82	20	56	82	17	34	53
G:	53	84	51	39	15	51	0	27	38	44	36	35	30	63	57	75	19	45	50	83
H:	27	60	77	54	66	53	57	0	62	79	32	74	47	21	19	16	28	69	52	82
I:	73	49	67	80	32	48	37	69	0	43	70	56	57	54	38	27	73	20	32	65
J:	55	64	28	73	41	28	72	49	41	0	59	38	74	47	26	21	17	77	53	71
K:	16	80	70	39	61	16	18	19	73	52	0	83	74	79	79	31	38	40	24	73
L:	61	54	47	71	76	27	24	73	37	40	18	0	42	82	33	23	68	78	75	17
M:	15	82	17	71	33	52	70	31	54	36	71	17	0	53	44	39	55	47	17	73
N:	16	65	73	21	30	45	25	15	80	82	39	81	74	0	73	19	82	65	56	16
O:	68	22	77	81	22	51	17	70	22	48	68	47	66	74	0	37	16	40	63	39
P:	52	64	50	77	30	72	57	51	22	46	38	22	19	26	43	0	45	43	61	50
Q:	41	27	48	84	29	26	78	81	32	52	40	44	47	64	41	27	0	81	46	19
R:	49	53	62	28	61	15	68	34	74	70	33	33	42	63	56	80	52	0	74	48
S:	19	38	44	63	42	22	72	41	80	28	15	81	44	34	70	84	22	19	0	65
T:	38	30	43	56	67	71	28	36	58	18	66	24	57	82	63	23	72	16	38	0

Rutas:

A->B->C->D->E->F->G->H->I->J->K->L->M->N->O->P->Q->R->S->T->	\$974
A->B->C->D->E->F->G->M->I->J->K->L->H->N->O->P->Q->R->S->T->	\$968
A->B->C->D->E->F->G->M->I->J->K->L->O->N->H->P->Q->R->S->T->	\$902
A->B->C->D->E->F->G->M->I->J->S->L->O->N->H->P->Q->R->K->T->	\$861
A->B->C->D->E->F->G->M->I->J->S->K->O->N->H->P->Q->R->L->T->	\$785
A->B->C->D->E->T->G->M->I->J->S->K->O->N->H->P->Q->R->L->F->	\$775
A->B->C->D->E->T->G->M->J->I->S->K->O->N->H->P->Q->R->L->F->	\$734
A->B->C->D->E->T->G->M->J->I->S->R->O->N->H->P->Q->K->L->F->	\$724
A->B->C->D->E->T->L->M->J->I->S->R->O->N->H->P->Q->K->G->F->	\$691
J->B->C->D->E->T->L->M->A->I->S->R->O->N->H->P->Q->K->G->F->	\$675
J->B->C->D->E->T->L->M->A->I->S->R->F->N->H->P->Q->K->G->O->	\$648
N->B->C->D->E->T->L->M->A->I->S->R->F->J->H->P->Q->K->G->O->	\$626
N->B->C->D->E->T->L->M->A->I->S->R->F->J->H->P->Q->O->G->K->	\$605
N->T->C->D->E->B->L->M->A->I->S->R->F->J->H->P->Q->O->G->K->	\$593
N->T->C->D->E->B->L->M->A->I->S->R->F->J->Q->P->H->O->G->K->	\$556
N->T->C->D->E->B->L->M->A->H->S->R->F->J->Q->P->I->O->G->K->	\$555
N->T->C->D->E->B->L->H->A->M->S->R->F->J->Q->P->I->O->G->K->	\$551
N->T->C->D->E->B->P->H->A->M->S->R->F->J->Q->L->I->O->G->K->	\$547
N->T->C->D->E->L->P->H->A->M->S->R->F->J->Q->B->I->O->G->K->	\$541
N->T->C->D->G->L->P->H->A->M->S->R->F->J->Q->B->I->O->E->K->	\$530
N->D->C->T->G->L->P->H->A->M->S->R->F->J->Q->B->I->O->E->K->	\$518
N->D->C->T->P->L->G->H->A->M->S->R->F->J->Q->B->I->O->E->K->	\$477

Ilustración 10: Ruta más corta

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            Random Azar = new();

            /* SUDOKU original, los ceros son los números
               que se deben hallar */
            int[,] Original = new int[9, 9] {
                {1, 0, 0, 0, 0, 7, 0, 9, 0},
                {0, 3, 0, 0, 2, 0, 0, 0, 8},
                {0, 0, 9, 6, 0, 0, 5, 0, 0},
                {0, 0, 5, 3, 0, 0, 9, 0, 0},
                {0, 1, 0, 0, 8, 0, 0, 0, 2},
                {6, 0, 0, 0, 0, 4, 0, 0, 0},
                {3, 0, 0, 0, 0, 0, 0, 1, 0},
                {0, 4, 0, 0, 0, 0, 0, 0, 7},
                {0, 0, 7, 0, 0, 0, 3, 0, 0}
            };

            /* Mantiene el ciclo hasta que resuelva el Sudoku */
            bool Finalizar;

            /* Lleva el número de iteraciones */
            int Ciclos = 0;

            /* Copia el SUDOKU original para trabajar en esta copia */
            int[,] Copia = new int[9, 9];

            /* Cada cuantos ciclos borra números para destrabar */
            int DESTRUYE = 700;

            /* Ciclo que llenará el sudoku completamente */
            do {
                /* Copia el sudoku original */
                for (int Fil = 0; Fil < 9; Fil++) {
                    for (int Col = 0; Col < 9; Col++)
                        if (Original[Fil, Col] != 0)
                            Copia[Fil, Col] = Original[Fil, Col];
                }

                bool numValido;
                int Fila, Columna, Numero;
                do {
                    /* Busca un número al azar para colocar en alguna celda */
                    Fila = Azar.Next(9); /* Una posición X de 0 a 8 */
                } while (!numValido);
            } while (!Finalizar);
        }
    }
}
```

```

Columna = Azar.Next(9); /* Una columna de 0 a 8 */
Numero = Azar.Next(1, 10); /* Un número al azar de 1 a 9 */
numValido = true;

/* Chequea si el número no se repite ni vertical
   ni horizontalmente */
for (int Cont = 0; Cont < 9; Cont++)
    if (Copia[Cont, Columna] == Numero ||
        Copia[Fila, Cont] == Numero)
        numValido = false;

} while (!numValido);

/* Si el número no se repite entonces valida que
   no se repita dentro del cuadro interno */
int cuadroFila = Fila - Fila % 3;
int cuadroColumna = Columna - Columna % 3;
for (int i = cuadroFila; i < cuadroFila + 3; i++) {
    for (int j = cuadroColumna; j < cuadroColumna + 3; j++) {
        if (Copia[i, j] == Numero) {
            numValido = false;
        }
    }
}

/* Si todo está bien, entonces se pone el número */
if (numValido)
    Copia[Fila, Columna] = Numero;

/* Chequea si se completó el sudoku completamente */
Finalizar = true;
for (Fila = 0; Fila < 9; Fila++)
    for (Columna = 0; Columna < 9; Columna++)
        if (Copia[Fila, Columna] == 0)
            Finalizar = false;

/* Cada cierto número de ciclos, para destrabar,
   borra la tercera parte de lo completado */
Ciclos++;
if (Ciclos % DESTROYE == 0)
    for (Fila = 0; Fila < 9; Fila++)
        for (Columna = 0; Columna < 9; Columna++)
            if (Azar.NextDouble() < 0.34)
                Copia[Fila, Columna] = 0;
} while (!Finalizar);

//Imprime el sudoku original
Console.WriteLine("Sudoku Original");

```

```

for (int Fila = 0; Fila < 9; Fila++) {
    for (int Columna = 0; Columna < 9; Columna++)
        if (Original[Fila, Columna] == 0)
            Console.Write("_ ");
        else
            Console.Write(Original[Fila, Columna] + " ");
    Console.WriteLine(" ");
}

//Imprime el sudoku resuelto
Console.WriteLine("\r\nCiclos totales usados: " + Ciclos);
Console.WriteLine("Sudoku Resuelto");
for (int Fila = 0; Fila < 9; Fila++) {
    for (int Columna = 0; Columna < 9; Columna++)
        Console.Write(Copia[Fila, Columna] + " ");
    Console.WriteLine(" ");
}
}
}
}

```

```
Consola de depuración de Mi X + v

Sudoku Original
1 _ _ _ 7 _ 9 _
_ 3 _ _ 2 _ _ 8
_ _ 9 6 _ _ 5 _
_ _ 5 3 _ _ 9 _
_ 1 _ _ 8 _ _ 2
6 _ _ _ 4 _ _ _
3 _ _ _ _ _ 1 _
_ 4 _ _ _ _ _ 7
_ _ 7 _ _ _ 3 _

Ciclos totales usados: 6035430
Sudoku Resuelto
1 6 2 8 5 7 4 9 3
5 3 4 1 2 9 6 7 8
7 8 9 6 4 3 5 2 1
4 7 5 3 1 2 9 8 6
9 1 3 5 8 6 7 4 2
6 2 8 7 9 4 1 3 5
3 5 6 4 7 8 2 1 9
2 4 1 9 3 5 8 6 7
8 9 7 2 6 1 3 5 4
```

Ilustración 11: Resuelve el Sudoku

El problema de las tres puertas

El problema de las tres puertas, también conocido como el problema de Monty Hall, es un acertijo matemático de probabilidad que se presenta en diferentes variantes. En una de las variantes, se tienen tres puertas etiquetadas como "coche", "cabra 1" y "cabra 2". Detrás de una de las puertas se encuentra un coche, mientras que detrás de las otras dos hay cabras. El objetivo del acertijo es determinar qué puerta contiene el coche, utilizando solo una elección de puerta y sin abrir ninguna de las puertas.

La mecánica del acertijo es la siguiente: el concursante debe elegir una puerta entre tres (todas cerradas); el premio consiste en llevarse lo que se encuentra detrás de la elegida. Se sabe con certeza que tras una de ellas se oculta un automóvil, y tras las otras dos hay cabras. Una vez que el concursante haya elegido una puerta y comunicado su elección a los presentes, el presentador, que sabe lo que hay detrás de cada puerta, abrirá una de las otras dos, en la que habrá una cabra. A continuación, le da la opción al concursante de cambiar, si lo desea, de puerta (tiene dos opciones). ¿Debe el concursante mantener su elección original o escoger la otra puerta? ¿Hay alguna diferencia?

La respuesta correcta es que el concursante debe cambiar de puerta. Si el concursante cambia de puerta, la probabilidad de ganar el coche es de $2/3$, mientras que, si mantiene su elección original, la probabilidad de ganar el coche es de $1/3$.

I/010.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            /* Generador de números pseudo-aleatorios */
            Random Azar = new();

            /* Contador de éxitos si el jugador
             cambia su decisión inicial */
            int CambiaDecision = 0;

            /* Contador de éxitos si el jugador
             insiste en mantener su decisión inicial */
            int InsisteNOCambiar = 0;

            for (int prueba = 1; prueba <= 1000000; prueba++) {
                /* Selecciona una caja al azar que será la ganadora */
                int cajaGanadora = Azar.Next(3);

                /* El jugador selecciona una caja al azar */
                int cajaJugador = Azar.Next(3);
```

```

    /* Si el jugador escogió por casualidad la
       caja ganadora, el dueño escoge al azar
       alguna de las dos cajas restantes */
    int cajaDueno;
    if (cajaGanadora == cajaJugador)
        do {
            cajaDueno = Azar.Next(3);
        } while (cajaDueno == cajaGanadora);
    else
        /* El dueño sólo puede escoger la caja
           que no tiene premio */
        do {
            cajaDueno = Azar.Next(3);
        } while (cajaDueno == cajaGanadora ||
            cajaDueno == cajaJugador);

    /* El jugador NO cambia su elección */
    if (cajaGanadora == cajaJugador)
        InsisteNOCambiar++;

    /* El jugador SI cambia su elección */
    int nuevaCaja;
    do {
        nuevaCaja = Azar.Next(3);
    } while (nuevaCaja == cajaDueno || nuevaCaja == cajaJugador);
    if (nuevaCaja == cajaGanadora) CambiaDecision++;
}

Console.WriteLine("Número de aciertos");
Console.WriteLine("SIN cambiar la elección: " + InsisteNOCambiar);
Console.WriteLine("CAMBIANDO la elección: " + CambiaDecision);
}
}
}

```

```

Consola de depuración de Mi
Número de aciertos
SIN cambiar la elección: 332465
CAMBIANDO la elección: 667535
C:\Users\engin\source\repos\Ejemplo\

```

Ilustración 12: Problema de las tres puertas

Área bajo la curva

El área bajo la curva es un concepto matemático que se utiliza para calcular el área de una región limitada por una curva y el eje x en un intervalo cerrado. Para encontrar el área bajo una curva, se utilizan integrales definidas. En general, el proceso para encontrar el área bajo una curva implica los siguientes pasos:

1. Formar una integral definida con la ecuación $Y=F(X)$ dada.
2. Obtener la integral de la función.
3. Se evalúan los límites superior e inferior en la expresión integrada.
4. Se obtiene la diferencia de los valores obtenidos del límite superior y límite inferior.

¿Pero en el caso que la integral definida sea muy compleja de resolver? En ese caso, se hace uso del siguiente método

Paso 1: Tiene la ecuación $Y=F(X)$

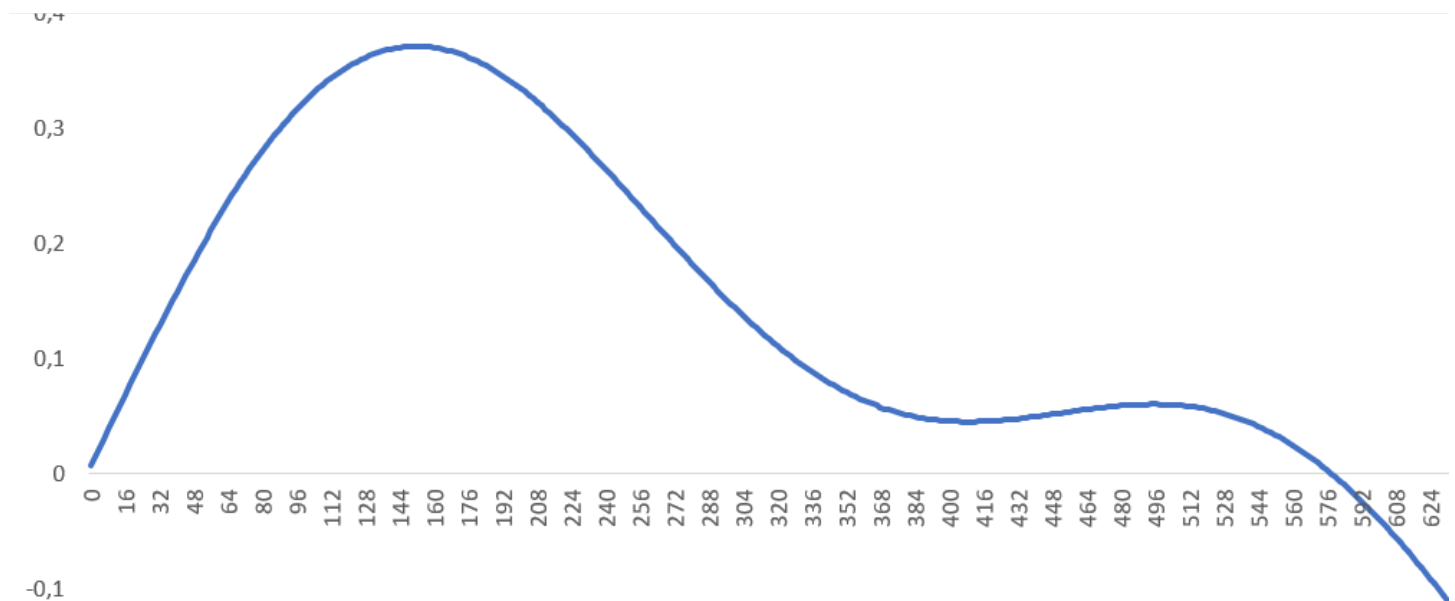


Ilustración 13: Curva $Y = F(X)$

Paso 2: Se dibujan los límites X mínimo y X máximo para calcular el área, con eso se tiene el tamaño de la base

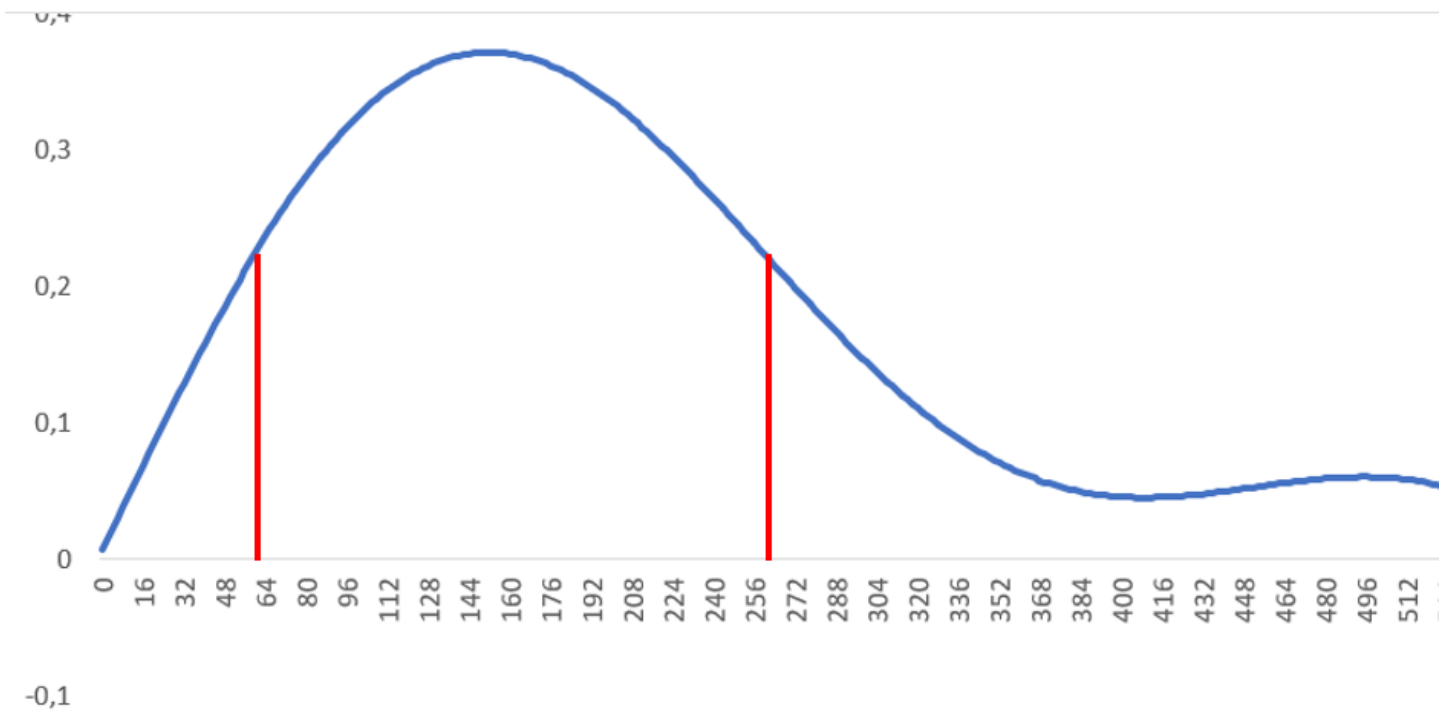


Ilustración 14: Los límites X_{min} y X_{max} para hallar el área interna

Paso 3: Se dibuja un rectángulo que contenga la curva en su interior. Luego ya se tiene el valor del tamaño de la base y la altura del rectángulo (que sería el mayor valor de Y entre X mínimo y Xmáximo)

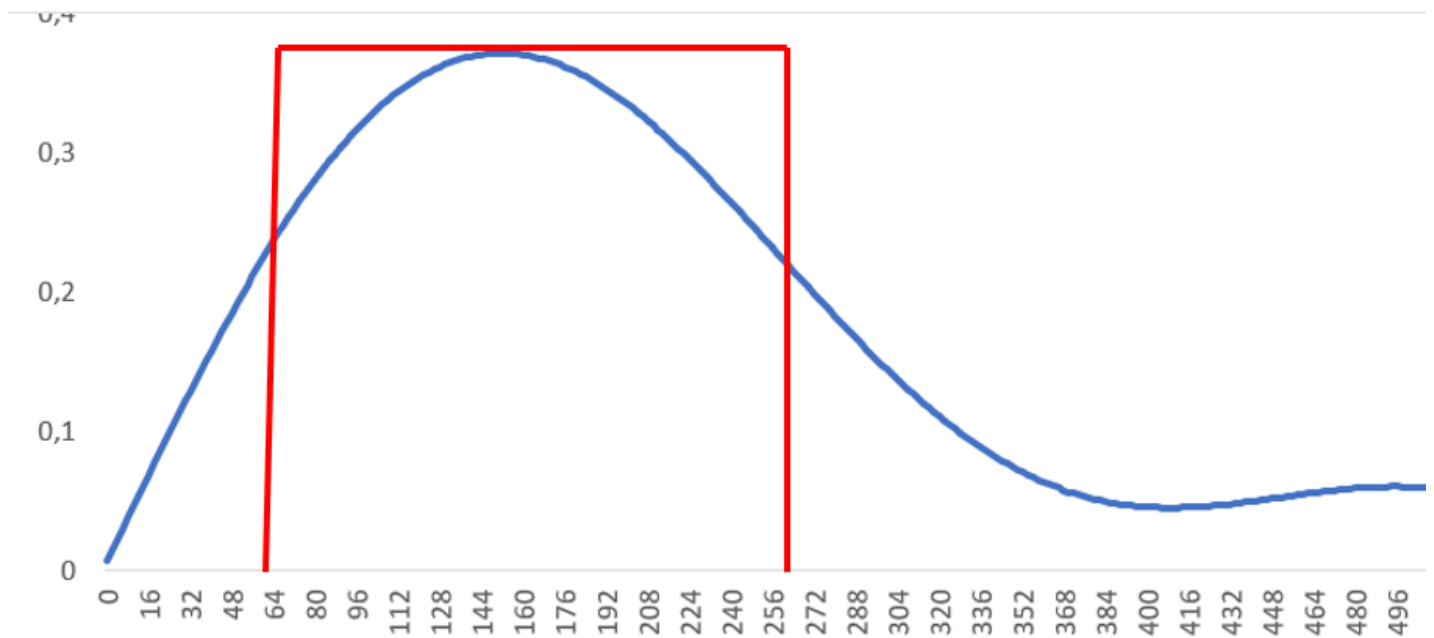


Ilustración 15: Se dibuja un rectángulo que cubra la curva

Paso 4: Se lanzan muchos puntos al azar al interior del rectángulo

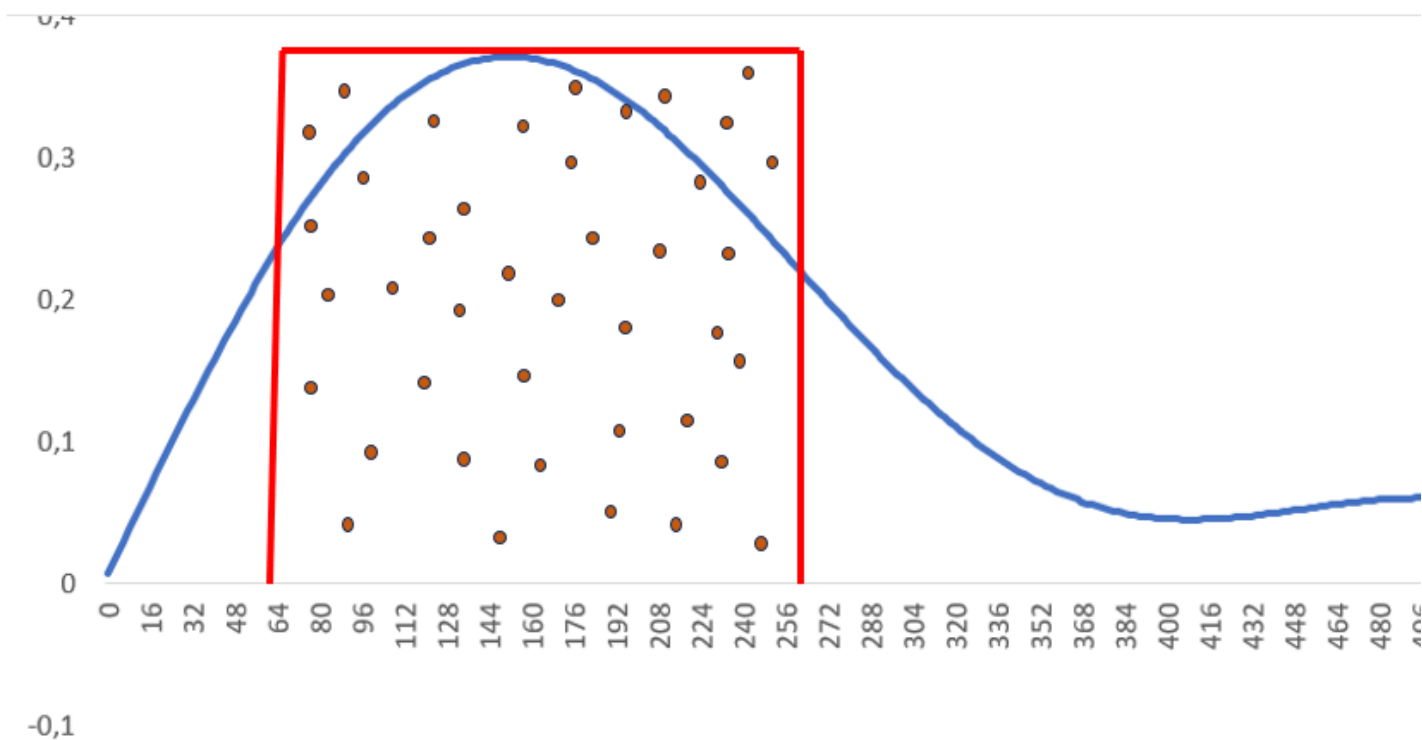


Ilustración 16: Se lanzan puntos al azar al interior del rectángulo

Paso 5: Se calcula el área bajo la curva con la siguiente ecuación:

$$\text{Área bajo la curva} = \text{Área del rectángulo} * \frac{\text{Total puntos al interior de la curva}}{\text{Total puntos lanzados}}$$

```

namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Área bajo la curva

            //Mínimo valor en X, máximo valor en X
            double MinValX = -5;
            double MaxValX = 5;

            //Validar que no hayan puntos de corte
            if (HayPuntosCorte(MinValX, MaxValX)) {
                Console.Write("Hay puntos de corte, ");
                Console.WriteLine("no es válido para hallar el área");
            }
            else {
                double Ymax = MaxMinY(MinValX, MaxValX);
                double Area = AreaC(MinValX, MaxValX, Ymax);
                Console.WriteLine("Área es: " + Area);
            }
        }

        //Retorna true si hay puntos de corte entre los valores
        //de X mínimo y máximo dados
        static public bool HayPuntosCorte(double MinX, double MaxX) {
            int Positivos = 0;
            int Negativos = 0;
            for (double X = MinX; X <= MaxX; X += 0.001) {
                double Y = Ecuacion(X);
                if (Y > 0) Positivos++;
                if (Y < 0) Negativos++;
            }
            if (Positivos != 0 && Negativos != 0) return true;
            return false;
        }

        //Retorna el mínimo valor de Y (área está por debajo del eje X)
        //o el máximo valor de Y (área está por encima del eje X)
        static public double MaxMinY(double MinX, double MaxX) {
            double MaximoY = double.MinValue;
            double MinimoY = double.MaxValue;
            bool Orienta = false;
            for (double X = MinX; X <= MaxX; X += 0.001) {
                double Y = Ecuacion(X);
                if (Y > 0) Orienta = true;
                if (Y > MaximoY) MaximoY = Y;
                if (Y < MinimoY) MinimoY = Y;
            }
        }
    }
}

```

```

        if (Orienta) return MaximoY;
        return MinimoY;
    }

    //Calcula el área bajo la curva usando el método Monte Carlo.
    //Siguiendo las directrices matemáticas,
    //el área será positiva si la curva está por encima del eje X,
    //el área sera negativa si la curva está por debajo del eje X
    static public double AreaC(double MinX, double MaxX, double Ymax) {
        Random Azar = new();
        int PuntosDentro = 0;
        int PuntosTotal = 7000000;
        for (int puntos = 1; puntos <= PuntosTotal; puntos++) {
            double Xazar = Azar.NextDouble() * (MaxX - MinX) + MinX;
            double Yazar = Azar.NextDouble() * Ymax;
            double Y = Ecuacion(Xazar);
            if (Yazar <= Y && Ymax > 0) PuntosDentro++;
            if (Yazar >= Y && Ymax < 0) PuntosDentro++;
        }
        double Valor = (double)PuntosDentro / PuntosTotal;
        double AreaTotal = (MaxX - MinX) * Ymax * Valor;
        return AreaTotal;
    }

    static public double Ecuacion(double X) {
        double Y = -1 * Math.Abs(Math.Cos(X) - Math.Sin(X) * X);
        return Y;
    }
}

```


Área es: -18,23635566437144

Ilustración 17: Área hallada

DE LOS CREADORES DE WOLFRAM LANGUAGE Y MATHEMATICA



integrate Y = -1 * abs(cos(X) - sin(X) * X) from X= -5 to 5

LENGUAJE NATURAL

ENTRADA MATEMÁTICA

TECLADO EXTENDIDO

EJEMPLOS

CARGAR

ALEATORIO

Integral definida

Más dígitos

Solución paso a paso

$$\int_{-5}^5 -|\cos(X) - \sin(X) X| dX =$$

$$-2 \left(\cos\left(\text{la raíz de } \cos(x) - x \sin(x) \text{ cercana a } x = -3,42562\right) \right.$$

$$\left. \cos\left(\text{la raíz de } \cos(x) - x \sin(x) \text{ cercana a } x = -0,860334\right) \right.$$

$$\left. \cos\left(\text{la raíz de } \cos(x) - x \sin(x) \text{ cercana a } x = 0,860334\right) \right.$$

$$\left. \cos\left(\text{la raíz de } \cos(x) - x \sin(x) \text{ cercana a } x = 3,42562\right) \right.$$

$$\left. \cos(5) \right) \approx -18,234$$

Ilustración 18: Comparando con un servicio externo

Dos robots se encuentran

En un arreglo bidimensional se ubican dos robots, cada uno en una casilla seleccionada al azar. No tienen sensores visuales, luego cada robot no sabe dónde está el otro robot. La simulación es averiguar en cuál escenario es más probable que haya un encuentro de los dos robots:

1. Un robot quieto mientras el otro se mueve al azar por el tablero.
2. Los dos robots moviéndose al tiempo al azar por el tablero.

I/012.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            /* Encuentro de dos robots sin sensor visual
               en un tablero.
               ¿Cuál es la mejor estrategia?
               1. Un robot quieto y el otro moviéndose.
               2. Los dos robots moviéndose.
            */

            Random Azar = new();

            //Tamaño del tablero
            int Filas = 50;
            int Columnas = 50;

            int Estral = 0;
            int Estra2 = 0;
            for (int Pruebas = 1; Pruebas <= 500; Pruebas++) {
                //Fila, Columna => Robot A
                int Fa = Azar.Next(Filas);
                int Ca = Azar.Next(Columnas);

                //Fila, Columna => Robot B
                int Fb, Cb;
                do {
                    Fb = Azar.Next(Filas);
                    Cb = Azar.Next(Columnas);
                } while (Fa == Fb || Ca == Cb);

                Estral += Estrategia1(Azar, Fa, Ca, Fb, Cb, Filas, Columnas);
                Estra2 += Estrategia2(Azar, Fa, Ca, Fb, Cb, Filas, Columnas);
            }

            Console.Write("Uno quieto y el otro moviéndose");
            Console.WriteLine(". Movimientos: " + Estral);
        }
    }
}
```

```

        Console.Write("Ambos robots se están moviendo.");
        Console.WriteLine("  Movimientos: " + Estra2);
    }

    static int Estrategia1(Random Azar, int Fa, int Ca, int Fb, int Cb,
        int Filas, int Columnas) {

        //Estrategia 1. Un robot quieto (B) y el otro moviéndose (A)
        int TotalMovimientos = 0;
        do {
            TotalMovimientos++;
            int FilM, ColM;
            do {
                FilM = Azar.Next(-1, 2);
                ColM = Azar.Next(-1, 2);
            } while (Fa + FilM < 0 || Fa + FilM >= Filas ||
                Ca + ColM < 0 || Ca + ColM >= Columnas ||
                (FilM == 0 && ColM == 0));
            Fa += FilM;
            Ca += ColM;
        } while (Fa != Fb || Ca != Cb);
        return TotalMovimientos;
    }

    static int Estrategia2(Random Azar, int Fa, int Ca, int Fb, int Cb,
        int Filas, int Columnas) {
        //Estrategia 2. Los dos robots moviéndose
        int TotalMovimientos = 0;
        do {
            TotalMovimientos++;

            //Mueve Robot A
            int A_FilM, A_ColM;
            do {
                A_FilM = Azar.Next(-1, 2);
                A_ColM = Azar.Next(-1, 2);
            } while (Fa + A_FilM < 0 || Fa + A_FilM >= Filas ||
                Ca + A_ColM < 0 || Ca + A_ColM >= Columnas ||
                (A_FilM == 0 && A_ColM == 0));
            Fa += A_FilM;
            Ca += A_ColM;

            //Mueve RobotB
            int B_FilM, B_ColM;
            do {
                B_FilM = Azar.Next(-1, 2);
                B_ColM = Azar.Next(-1, 2);
            } while (Fb + B_FilM < 0 || Fb + B_FilM >= Filas ||

```

```

        Cb + B_ColM < 0 || Cb + B_ColM >= Columnas ||
        (B_FilM == 0 && B_ColM == 0));
    Fb += B_FilM;
    Cb += B_ColM;

    } while (Fa != Fb || Ca != Cb);
    return TotalMovimientos;
}
}
}

```

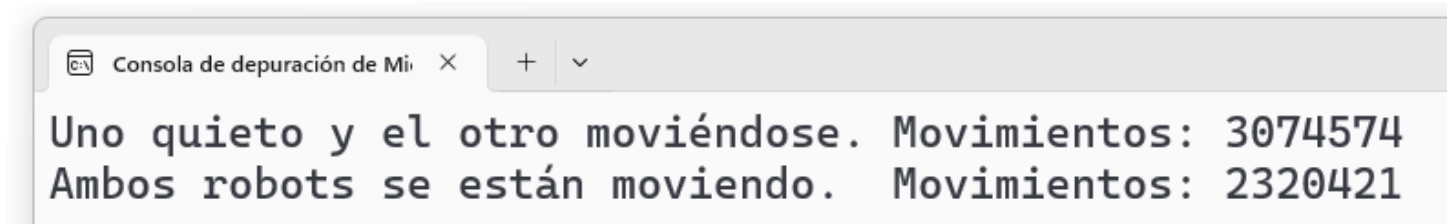


Ilustración 19: Dos robots moviéndose

Simulación de Inventarios

Tener un inventario genera costos. Luego la meta es disminuirlos.

Para simular el comportamiento de un inventario, se empieza por algo sencillo: es el inventario de un solo producto en particular.

Un inventario tiene lo siguiente:

1. Cada unidad del producto en el inventario genera un costo diario de inventario. Por ejemplo: si tiene almacenadas 400 unidades y cada unidad genera un costo de \$30, entonces, el costo de ese día es de $400 * \$30 = \12000 . Nota: Ese costo se genera al final del día con las unidades que quedan en el inventario.
2. Una demanda D variable diaria de unidades de ese producto que va disminuyendo el número de productos del inventario. Por ejemplo: hay 700 unidades y ese día la demanda fue de 80 unidades, luego al final del día habría $700 - 80 = 620$ unidades.
3. Si la demanda de ese día supera lo que había en el inventario, se entrega lo que queda del inventario, pero al no poder suplir a la demanda se paga por costo de imagen (se pierde credibilidad y eso es medible). Por ejemplo: hay 150 unidades y la demanda fue de 210 unidades. Se logra suplir 150, pero el resto $210 - 150 = 60$ unidades se pagan por pérdida de imagen. El costo de pérdida de imagen se calcula así: unidades que no se pudo suplir * costo de pérdida de imagen por unidad que no fue posible suplir. Supongamos que ese costo por unidad que no fue posible vender es de \$950. Luego la pérdida de imagen es de $60 * \$950 = \57.000
4. Si quien gestiona el inventario, ve que el inventario está por debajo de un punto R ese día, hace el pedido al proveedor de ese producto, la cantidad a pedir es Q . El sólo hecho de hacer el pedido genera un costo de pedido $\$CP$. Cabe anotar que es hasta el final del día que no se sabe si el inventario está igual o por debajo de R . Si al finalizar el día es menor o igual a R , entonces se hace el pedido, pero los días de entrega de ese pedido cuentan a partir del siguiente día, un ejemplo: al final del lunes se hace el pedido y tarda 3 días (esperaría martes, miércoles, jueves) y es hasta el viernes que recibe el pedido a primera hora.
5. El proveedor tardará un tiempo T variable de días para llegar con ese pedido. Mientras tanto, quien gestiona el inventario tendrá que defenderse con lo que le queda del inventario. No se pueden hacer más pedidos hasta que llegue el proveedor. El proveedor siempre llega con el pedido justo antes de abrir el inventario el día de la entrega. Por ejemplo: el día amanece con 175 unidades, el proveedor llega con el pedido Q de 400 unidades, luego cuando abra el inventario tendrá $175 + 400 = 575$ unidades.

El archivo de Excel "Inventario.xlsx" muestra la simulación del comportamiento del inventario.

B5									
	A	B	C	D	E	F	G	H	I
1									
2				ENTRADAS DEL MODELO DE SIMULACIÓN					
3									
4									
5	R	120				Unidades compradas diarias por los clientes	Probabilidad	Minimo	Maximo
6	Q	240				25	10%	0	0.1
7	Inventario Inicial	570				26	20%	0.1	0.3
8	Costo de pérdida de prestigio por unidad no entregada al cliente	\$ 400				27	30%	0.3	0.6
9	Costo de hacer el pedido	\$ 7,000				28	25%	0.6	0.85
10	Costo de almacenar una unidad por día	\$ 125				29	15%	0.85	1
11									
12						Tiempo en días para que el proveedor entreque el pedido	Probabilidad	Minimo	Maximo
13						3	20%	0	0.2
14						4	30%	0.2	0.5
15						5	35%	0.5	0.85
16						6	10%	0.85	0.95
17						7	5%	0.95	1
18									

Ilustración 20: Archivo en Excel con las entradas del modelo de simulación

	L	M	N	O	P	Q
1	Simulación de Inventario de punto RQ					
2						
3						
4						
5	Generador congruencial lineal de números aleatorios #1					
6	A	X0	B	N		
7	2678917	4579991	1317513	9824217		
8						
9						
10						
11	Generador congruencial lineal de números aleatorios #2					
12	A	X0	B	N		
13	7921083	6731297	9021531	9420811		
14						

Ilustración 21: Semillas de los generadores de números pseudo-aleatorios

K1031										
	A	B	C	D	E	F	G	H	I	J
20				RESULTADOS O SALIDAS DEL MODELO DE SIMULACIÓN						
21										
22										
23	Costo Total	Costo Inventario	Costo Ordenar	Costo pérdida de prestigio						
24	\$ 14,675,475	\$ 12,940,875	\$ 707,000	\$ 1,027,600						
25										
26										
27										
28	Día	#Azar	Demanda	Inventario	Costo Inventario	Costo Ordenar	#Azar	Llega Pedido	Cuenta atrás pedido	Costo pérdida de prestigio
29	0			570					-1	
30	1	0.71211436	28	542	\$ 67,750	\$ 0	-1.000	0	-1	
31	2	0.398659761	27	515	\$ 64,375	\$ 0	-1.000	0	-1	
32	3	0.544731555	27	488	\$ 61,000	\$ 0	-1.000	0	-1	
33	4	0.756398194	28	460	\$ 57,500	\$ 0	-1.000	0	-1	
34	5	0.115803326	26	434	\$ 54,250	\$ 0	-1.000	0	-1	
35	6	0.6317107	28	406	\$ 50,750	\$ 0	-1.000	0	-1	
36	7	0.667960001	28	378	\$ 47,250	\$ 0	-1.000	0	-1	
37	8	0.536886349	27	351	\$ 43,875	\$ 0	-1.000	0	-1	
38	9	0.102354315	26	325	\$ 40,625	\$ 0	-1.000	0	-1	
39	10	0.848191159	28	297	\$ 37,125	\$ 0	-1.000	0	-1	
40	11	0.848249789	28	269	\$ 33,625	\$ 0	-1.000	0	-1	
41	12	0.9148325	29	240	\$ 30,000	\$ 0	-1.000	0	-1	
42	13	0.470878646	27	213	\$ 26,625	\$ 0	-1.000	0	-1	

Ilustración 22: Comportamiento día a día de la simulación

Se replica esa simulación en C# con una aplicación de consola.

```
Costo total inventario;12940875
Costo total ordenar;707000
Costo total prestigio;1027600
Costo total;14675475
```

Ilustración 23: Programa en C# que replica el comportamiento mostrado en Excel


```

namespace Ejemplo {

    //Generador congruencia lineal de números
    //pseudo-aleatorios
    internal class Aleatorio {
        private long A, B, X0, N;
        public Aleatorio(long A, long X0, long B, long N) {
            this.A = A;
            this.X0 = X0;
            this.B = B;
            this.N = N;
        }

        public double Numero() {
            X0 = (A * X0 + B) % N;
            return (double) X0 / N;
        }
    }
}

```

```

namespace Ejemplo {

    //Demanda diaria del producto
    internal class Demanda {
        private int Compra;
        private int Probabilidad;

        public Demanda(int Compra, int Probabilidad) {
            this.Compra = Compra;
            this.Probabilidad = Probabilidad;
        }

        public int getCompra() { return Compra; }
        public int getProbabilidad() { return Probabilidad; }
    }
}

```

```

namespace Ejemplo {

    //Los días que tarda el proveedor en entregar el producto
    //desde el pedido
    internal class Proveedor {
        int Dias;
        int Probabilidad;

        public Proveedor(int Dias, int Probabilidad) {
            this.Dias = Dias;
            this.Probabilidad = Probabilidad;
        }

        public int getDias() { return Dias; }
        public int getProbabilidad() { return Probabilidad; }
    }
}

```

```

namespace Ejemplo {

    //Simula el día a día del inventario de punto RQ
    internal class Simular {
        int R, Q;
        int Inicial;
        int CostoPerdida;
        int CostoPedido;
        int CostoAlmacena;
        List<Demanda> DemandaCliente;
        List<Proveedor> DiasProveedor;
        int DiasSimula;
        Aleatorio AzarA;
        Aleatorio AzarB;

        public void Parametros() {
            /* Punto de Reorden.Cuando el inventario al final del día queda
             * igual o por debajo de este valor, entonces se hace el pedido
             * al proveedor */
            R = 120;

            /* Esta es la cantidad fija que se le pide al proveedor */
            Q = 240;
        }
    }
}

```

```

    /* Inventario Inicial */
    Inicial = 570;

    /* Costo de pérdida de prestigio por unidad no entregada al
cliente */
    CostoPerdida = 400;

    /* Costo de hacer el pedido */
    CostoPedido = 7000;

    /* Costo de almacenar una unidad por día */
    CostoAlmacena = 125;

    /* Comportamiento de la demanda,
    * las probabilidades deben sumar 100 */
    DemandaCliente = new List<Demanda>();
    DemandaCliente.Add(new Demanda(25, 10));
    DemandaCliente.Add(new Demanda(26, 20));
    DemandaCliente.Add(new Demanda(27, 30));
    DemandaCliente.Add(new Demanda(28, 25));
    DemandaCliente.Add(new Demanda(29, 15));

    /* Dias de entrega del proveedor,
    * las probabilidades deben sumar 100 */
    DiasProveedor = new List<Proveedor>();
    DiasProveedor.Add(new Proveedor(3, 20));
    DiasProveedor.Add(new Proveedor(4, 30));
    DiasProveedor.Add(new Proveedor(5, 35));
    DiasProveedor.Add(new Proveedor(6, 10));
    DiasProveedor.Add(new Proveedor(7, 5));

    /* Días a simular */
    DiasSimula = 1000;

    /* Generadores de números aleatorios */
    AzarA = new Aleatorio(2678917, 4579991, 1317513, 9824217);
    AzarB = new Aleatorio(7921083, 6731297, 9021531, 9420811);
}

/* Retorna la demanda de unidades dependiendo del
* del número aleatorio */

public int Demanda(double Valor) {
    int Acumula = 0;
    int Tope = (int)(Valor * 100);
    int Cont;
    for (Cont = 0; Cont < DemandaCliente.Count; Cont++) {

```

```

        Acumula += DemandaCliente[Cont].getProbabilidad();
        if (Acumula > Tope) break;
    }
    return DemandaCliente[Cont].getCompra();
}

/* Retorna los días dependiendo del número aleatorio */
public int Dias(double Valor) {
    int Acumula = 0;
    int Tope = (int) (Valor * 100);
    int Cont;
    for (Cont = 0; Cont < DiasProveedor.Count; Cont++) {
        Acumula += DiasProveedor[Cont].getProbabilidad();
        if (Acumula > Tope) break;
    }
    return DiasProveedor[Cont].getDias();
}

/* Proceso de simulación */
public void Proceso() {
    //Día cero
    AzarA.Numero();
    AzarB.Numero();

    //Inicia el día cero con un inventario inicial
    int Inventario = Inicial;

    //Días para llegar el pedido del proveedor
    int DiasLlegaPedido = 0, CuentaAtrasPedido = -1;

    //Costos diarios
    int CostoInventario, CostoOrdenar, CostoPrestigio;

    //Acumula Costos
    int AcumCostoInventario = 0, AcumCostoOrdenar = 0,
    AcumCostoPrestigio = 0;

    Console.WriteLine("Dia;Azar;Demanda;Inventario;Costo
    Inventario;Costo Ordenar;Azar;Llega Pedido;Cuenta atrás pedido;Costo
    pérdida de prestigio");

    for (int dias = 1; dias <= DiasSimula; dias++) {

        //Cuenta Atrás para llegar el pedido del proveedor
        if (DiasLlegaPedido > 0)
            CuentaAtrasPedido = DiasLlegaPedido;
        else {
            if (CuentaAtrasPedido > 0)

```

```

        CuentaAtrasPedido--;
    else
        CuentaAtrasPedido = -1;
}

//Valores aleatorios del día
double NumA = AzarA.Numero();
double NumB = AzarB.Numero();

//Valor del inventario
int NuevoValor = 0;
if (Inventario > 0) NuevoValor = Inventario;

//Compra de los clientes
int Compra = Demanda(NumA);
NuevoValor -= Compra;

//Si llega el pedido
if (CuentaAtrasPedido == 0) NuevoValor += Q;

//El nuevo valor de inventario
Inventario = NuevoValor;

//Costo del inventario
if (Inventario > 0)
    CostoInventario = Inventario * CostoAlmacena;
else
    CostoInventario = 0;

//Costo ordenar
CostoOrdenar = 0;
if (Inventario <= R && CuentaAtrasPedido <= 0)
    CostoOrdenar = CostoPedido;

//Segundo Azar si hay pedido
double SegundoAzar = -1;
if (CostoOrdenar > 0)
    SegundoAzar = NumB;

//Llega Pedido
DiasLlegaPedido = 0;
if (SegundoAzar > 0)
    DiasLlegaPedido = Dias(SegundoAzar);

//Pérdida de prestigio
CostoPrestigio = 0;
if (Inventario < 0)
    CostoPrestigio = Inventario * CostoPerdida * -1;

```

```

        //Imprime el día
        Console.WriteLine(dias + ";" + NumA + ";" + Compra + ";" +
Inventario + ";" + CostoInventario + ";" + CostoOrdenar + ";" +
SegundoAzar + ";" + DiasLlegaPedido + ";" + CuentaAtrasPedido + ";" +
CostoPrestigio);

        //Acumula costos
        AcumCostoInventario += CostoInventario;
        AcumCostoOrdenar += CostoOrdenar;
        AcumCostoPrestigio += CostoPrestigio;
    }

    //Imprime los costos totales
    Console.WriteLine("Costo total inventario;" +
AcumCostoInventario);
    Console.WriteLine("Costo total ordenar;" + AcumCostoOrdenar);
    Console.WriteLine("Costo total prestigio;" + AcumCostoPrestigio);

    //Costo total
    int CostoTotal = AcumCostoInventario + AcumCostoOrdenar +
AcumCostoPrestigio;
    Console.WriteLine("Costo total;" + CostoTotal);
}
}
}

```

I/013.7z

```

namespace Ejemplo {
    //Inventario
    class Program {
        static void Main() {
            Simular simular = new();
            simular.Parametros();
            simular.Proceso();
        }
    }
}

```

Hay que considerar que una simulación es sensible a los valores iniciales, en el caso de esta simulación, es el inventario inicial. Luego, se debería escoger un valor de inventario inicial muy cercano a Q (pedido) o hacer los totales de costos a partir de varios días, por ejemplo, del 100 en adelante, cuando se considera que el modelo se ha estabilizado.

Simulación de Inventarios: Buscando el mejor valor

En la simulación anterior, la decisión recae en los valores R y Q, en búsqueda de disminuir el costo total. Una técnica es probar sistemáticamente valores R y Q hasta dar con aquellos que obtengan el mínimo costo. Para esto, se modifica el programa anterior para que se dedique a calcular, no imprimir y además retorne el costo total. Importante: Debido a que las simulaciones son sensibles a los valores iniciales (en este caso, al valor del inventario inicial) y tanto Q como R tienen diversos valores, entonces los totales se calculan a partir de varios días de simulación para que esta se estabilice, en el código, a partir del día 500 (se puede variar).

I/014.7z

```
namespace Ejemplo {

    //Generador congruencia lineal de números
    //pseudo-aleatorios
    internal class Aleatorio {
        private long A, B, X0, N;
        public Aleatorio(long A, long X0, long B, long N) {
            this.A = A;
            this.X0 = X0;
            this.B = B;
            this.N = N;
        }

        public double Numero() {
            X0 = (A * X0 + B) % N;
            return (double) X0 / N;
        }
    }
}
```

I/014.7z

```
namespace Ejemplo {

    //Demanda diaria del producto
    internal class Demanda {
        private int Compra;
        private int Probabilidad;

        public Demanda(int Compra, int Probabilidad) {
            this.Compra = Compra;
            this.Probabilidad = Probabilidad;
        }
    }
}
```

```

    }

    public int getCompra() { return Compra; }
    public int getProbabilidad() { return Probabilidad; }
}
}

```

I/014.7z

```

namespace Ejemplo {

    //Los días que tarda el proveedor en entregar el producto
    //desde el pedido
    internal class Proveedor {
        int Dias;
        int Probabilidad;

        public Proveedor(int Dias, int Probabilidad) {
            this.Dias = Dias;
            this.Probabilidad = Probabilidad;
        }

        public int getDias() { return Dias; }
        public int getProbabilidad() { return Probabilidad; }
    }
}

```

I/014.7z

```

namespace Ejemplo {

    //Simula el día a día del inventario de punto RQ
    internal class Simular {
        int R, Q;
        int Inicial;
        int CostoPerdida;
        int CostoPedido;
        int CostoAlmacena;
        List<Demanda> DemandaCliente;
        List<Proveedor> DiasProveedor;
        int DiasSimula;
        int IniciaDiaCostoAcumula;
        Aleatorio AzarA;
        Aleatorio AzarB;
    }
}

```



```

public void Parametros(int R, int Q) {
    /* Punto de Reorden.Cuando el inventario al final
     * del día queda igual o por debajo de este valor,
     * entonces se hace el pedido al proveedor */
    this.R = R;

    /* Esta es la cantidad fija que se le pide al proveedor */
    this.Q = Q;

    /* Inventario Inicial */
    Inicial = 570;

    /* Costo de pérdida de prestigio por
     * unidad no entregada al cliente */
    CostoPerdida = 400;

    /* Costo de hacer el pedido */
    CostoPedido = 7000;

    /* Costo de almacenar una unidad por día */
    CostoAlmacena = 125;

    /* Comportamiento de la demanda,
     * las probabilidades deben sumar 100 */
    DemandaCliente = new List<Demanda>();
    DemandaCliente.Add(new Demanda(25, 10));
    DemandaCliente.Add(new Demanda(26, 20));
    DemandaCliente.Add(new Demanda(27, 30));
    DemandaCliente.Add(new Demanda(28, 25));
    DemandaCliente.Add(new Demanda(29, 15));

    /* Dias de entrega del proveedor,
     * las probabilidades deben sumar 100 */
    DiasProveedor = new List<Proveedor>();
    DiasProveedor.Add(new Proveedor(3, 20));
    DiasProveedor.Add(new Proveedor(4, 30));
    DiasProveedor.Add(new Proveedor(5, 35));
    DiasProveedor.Add(new Proveedor(6, 10));
    DiasProveedor.Add(new Proveedor(7, 5));

    /* Días a simular */
    DiasSimula = 2000;

    /* A partir de que día se acumulan los costos.
     * La razón es evitar que el valor inicial del
     * inventario, afecte a la simulación. Hay que
     * esperar que se estabilice */

```

```

        IniciaDiaCostoAcumula = 500;

        /* Generadores de números aleatorios */
        AzarA = new Aleatorio(2678917, 4579991, 1317513, 9824217);
        AzarB = new Aleatorio(7921083, 6731297, 9021531, 9420811);
    }

    /* Retorna la demanda de unidades dependiendo del
    * del número aleatorio */

    public int Demanda(double Valor) {
        int Acumula = 0;
        int Tope = (int) (Valor * 100);
        int Cont;
        for (Cont = 0; Cont < DemandaCliente.Count; Cont++) {
            Acumula += DemandaCliente[Cont].getProbabilidad();
            if (Acumula > Tope) break;
        }
        return DemandaCliente[Cont].getCompra();
    }

    /* Retorna los días dependiendo del número aleatorio */
    public int Dias(double Valor) {
        int Acumula = 0;
        int Tope = (int) (Valor * 100);
        int Cont;
        for (Cont = 0; Cont < DiasProveedor.Count; Cont++) {
            Acumula += DiasProveedor[Cont].getProbabilidad();
            if (Acumula > Tope) break;
        }
        return DiasProveedor[Cont].getDias();
    }

    /* Proceso de simulación */
    public int Proceso() {
        //Día cero
        AzarA.Numero();
        AzarB.Numero();

        //Inicia el día cero con un inventario inicial
        int Inventario = Inicial;

        //Días para llegar el pedido del proveedor
        int DiasLlegaPedido = 0, CuentaAtrasPedido = -1;

        //Costos diarios
        int CostoInventario, CostoOrdenar, CostoPrestigio;
    }

```

```

//Acumula Costos
int AcumCostoInventario = 0, AcumCostoOrdenar = 0;
int AcumCostoPrestigio = 0;

for (int dias = 1; dias <= DiasSimula; dias++) {

    //Cuenta Atrás para llegar el pedido del proveedor
    if (DiasLlegaPedido > 0)
        CuentaAtrasPedido = DiasLlegaPedido;
    else {
        if (CuentaAtrasPedido > 0)
            CuentaAtrasPedido--;
        else
            CuentaAtrasPedido = -1;
    }

    //Valores aleatorios del día
    double NumA = AzarA.Numero();
    double NumB = AzarB.Numero();

    //Valor del inventario
    int NuevoValor = 0;
    if (Inventario > 0) NuevoValor = Inventario;

    //Compra de los clientes
    int Compra = Demanda(NumA);
    NuevoValor -= Compra;

    //Si llega el pedido
    if (CuentaAtrasPedido == 0) NuevoValor += Q;

    //El nuevo valor de inventario
    Inventario = NuevoValor;

    //Costo del inventario
    if (Inventario > 0)
        CostoInventario = Inventario * CostoAlmacena;
    else
        CostoInventario = 0;

    //Costo ordenar
    CostoOrdenar = 0;
    if (Inventario <= R && CuentaAtrasPedido <= 0)
        CostoOrdenar = CostoPedido;

    //Segundo Azar si hay pedido
    double SegundoAzar = -1;

```

```

        if (CostoOrdenar > 0)
            SegundoAzar = NumB;

        //Llega Pedido
        DiasLlegaPedido = 0;
        if (SegundoAzar > 0)
            DiasLlegaPedido = Dias(SegundoAzar);

        //Pérdida de prestigio
        CostoPrestigio = 0;
        if (Inventario < 0)
            CostoPrestigio = Inventario * CostoPerdida * -1;

        //Acumula costos
        if (dias >= IniciaDiaCostoAcumula) {
            AcumCostoInventario += CostoInventario;
            AcumCostoOrdenar += CostoOrdenar;
            AcumCostoPrestigio += CostoPrestigio;
        }
    }

    //Costo total
    int CostoTotal = AcumCostoInventario;
    CostoTotal += AcumCostoOrdenar;
    CostoTotal += AcumCostoPrestigio;
    return CostoTotal;
}
}
}

```

I/014.7z

```

namespace Ejemplo {
    //Inventario
    class Program {
        static void Main() {
            int CostoTotal = int.MaxValue;

            int MejorR = -1, MejorQ = -1;
            for (int R = 1; R <= 1000; R++)
                for (int Q = 1; Q <= 1000; Q++) {
                    int Costo = SimularInventario(R, Q);
                    if (Costo < CostoTotal) {
                        CostoTotal = Costo;
                        MejorR = R;
                        MejorQ = Q;
                    }
                }
        }
    }
}

```

```

        Console.WriteLine("Costo: " + CostoTotal + " R = " +
MejorR + " Q = " + MejorQ);
    }
}

Console.WriteLine("Costo más bajo: " + CostoTotal);
Console.WriteLine("R = " + MejorR);
Console.WriteLine("Q = " + MejorQ);
}

static public int SimularInventario(int R, int Q) {
    Simular simular = new();
    simular.Parametros(R, Q);
    return simular.Proceso();
}
}
}

```

```

Costo: 18110400 R = 1 Q = 1
Costo: 18002400 R = 1 Q = 2
Costo: 17894400 R = 1 Q = 3
Costo: 17786400 R = 1 Q = 4
Costo: 17678400 R = 1 Q = 5
Costo: 17570400 R = 1 Q = 6
Costo: 17462400 R = 1 Q = 7
Costo: 17354400 R = 1 Q = 8
Costo: 17246400 R = 1 Q = 9
Costo: 17138400 R = 1 Q = 10
Costo: 17030400 R = 1 Q = 11
Costo: 16922400 R = 1 Q = 12
Costo: 16814400 R = 1 Q = 13

```

Ilustración 24: Inicia la simulación

```

Costo: 12017900 R = 78 Q = 103
Costo: 12012025 R = 79 Q = 104
Costo: 12008275 R = 80 Q = 105
Costo: 11996350 R = 82 Q = 107
Costo más bajo: 11996350
R = 82
Q = 107

```

Ilustración 25: Finaliza la simulación

Después de varios segundos (o minutos, dependiendo de la velocidad del PC), se obtienen los valores R y Q que generan el menor costo posible.

Jugadores y apuestas

Se da la siguiente situación: un grupo de jugadores, cada uno con un mismo capital inicial, empieza una serie de rondas de apuestas. Cada jugador apuesta N% de su capital, y ese N% es lo que puede perder o puede ganar. Por ejemplo, un jugador tiene \$100.000 de capital y apuesta el 15%, es decir, \$15.000; si gana, le suma \$15.000 a su capital, si pierde, le resta \$15.000 a su capital. ¿Y Cómo es el juego? Se lanza un número entero al azar entre 1 y 100, si ese número es menor o igual a N, el jugador pierde, caso contrario, gana. ¿Cuál sería el mejor N para aumentar el capital?

I/015.7z

```
namespace Ejemplo {
    internal class Apostador {
        public int Dinero;
        public int Riesgo;

        public Apostador(int dinero, int riesgo) {
            Dinero = dinero;
            Riesgo = riesgo;
        }
    }
}
```

I/015.7z

```
namespace Ejemplo {
    internal class Poblacion {
        int TotalJugadores;
        int DineroInicial;
        int DineroMinimo;
        int TotalCiclos;
        List<Apostador> Jugador;

        public void Parametros() {
            TotalJugadores = 700;

            //Con cuánto dinero arranca cada jugdor
            DineroInicial = 20000;

            //Si el capital de un jugador cae por debajo
            //de este número, se retira
            DineroMinimo = 1500;

            TotalCiclos = 4000;
            Jugador = [];
        }
    }
}
```

```

public void Proceso() {
    Random Azar = new();

    //Crea la población
    for (int cont = 1; cont <= TotalJugadores; cont++)
        Jugador.Add(new Apostador(DineroInicial, Azar.Next(1, 101)));

    //Ciclo de juegos
    for (int Ciclo = 1; Ciclo <= TotalCiclos; Ciclo++) {
        int num = 0;
        while (num < Jugador.Count) {

            //Si el jugador queda con capital por debajo
            //de DineroMinimo, se elimina
            if (Jugador[num].Dinero < DineroMinimo) {
                Jugador.RemoveAt(num);
                continue;
            }

            //Se enfrenta a una apuesta
            int Ganancia = Jugador[num].Dinero * Jugador[num].Riesgo /
100;

            if (Azar.Next(1, 101) <= Jugador[num].Riesgo)
                Jugador[num].Dinero -= Ganancia;
            else
                Jugador[num].Dinero += Ganancia;

            //Siguiente jugador
            num++;
        }

        //¿La población quedó en cero?
        if (Jugador.Count == 0) {
            Console.WriteLine("Colapso de la población");
            Environment.Exit(0);
        }

        //Si la población está por debajo de TotalJugadores,
        //entonces se busca el que más dinero tiene y se le
        //sacan copias modificadas
        if (Jugador.Count < TotalJugadores) {

            //Busca el que tiene más dinero
            int MaximoDinero = int.MinValue;
            int Cual = -1;
            for (num = 0; num < Jugador.Count; num++) {
                if (Jugador[num].Dinero > MaximoDinero) {

```



```

        MaximoDinero = Jugador[num].Dinero;
        Cual = num;
    }
}

//Saca copias modificadas del individuo con más dinero
while (Jugador.Count < TotalJugadores) {
    int CambiaRiesgo = 1;
    if (Azar.NextDouble() < 0.5) CambiaRiesgo = -1;
    int NuevoRiesgo = Jugador[Cual].Riesgo + CambiaRiesgo;
    Jugador.Add(new Apostador(DineroInicial, NuevoRiesgo));
}
}

Console.WriteLine("Riesgo promedio: ");
int Acumula = 0;
for (int num = 0; num < Jugador.Count; num++)
    Acumula += Jugador[num].Riesgo;
double Promedio = (double) Acumula / Jugador.Count;
Console.WriteLine(Promedio);
}
}
}

```

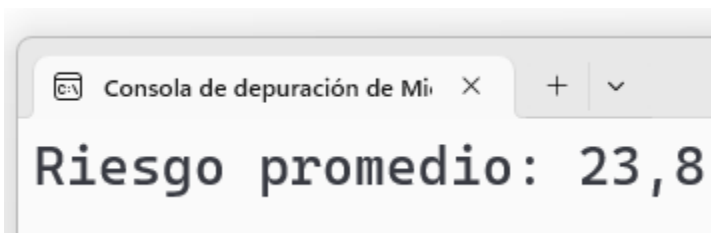


Ilustración 26: Valor en porcentaje a apostar

El valor de N en el programa es la variable "Riesgo", luego según la simulación con esos valores de dinero inicial (capital inicial), informa que N debe ser del 23.8% para obtener las mayores ganancias posibles.