SIMULACIÓN DE SISTEMAS

Práctica 2:

Modelos de Monte Carlo. Generadores de datos

Curso 2020/2021

Capítulo 1

Mi Segundo Modelo de Simulación de Monte Carlo

1.1. Planteamiento del Problema: Quiosco de Periódicos

Un establecimiento (por ejemplo un quiosco de periódicos) se abastece diariamente de un cierto producto, y necesita decidir cuántas unidades de ese producto pedir cada día. Este valor, s, se fija en un contrato a largo plazo con el proveedor (y una vez fijado no puede cambiarse). El establecimiento obtiene una ganancia de x euros por cada unidad vendida, y una pérdida de y euros por unidad no vendida al finalizar el día. La demanda D (o sea, el número de unidades del producto que se solicitan cada día) varía diariamente, pero se ha estudiado que se ajusta a una determinada distribución de probabilidad $P(D = d)$, que se detallará después. Se desea encontrar el valor óptimo de s, donde el criterio de optimalidad es maximizar la ganancia esperada.

1.2. Modelización Analítica

En términos más formales, el problema puede describirse de la siguiente forma:

Sea $g(s, x, y, d)$ la función de ganancia diaria cuando la demanda de ese día es $D = d$ (y supuesto que el número de unidades de producto pedidas al proveedor es s, y la ganancia y pérdida por unidad son x e y, respectivamente). Entonces

$$g(s, x, y, d) = \begin{cases} x * s & \text{si } d \geq s \\ \\ x * d - (s - d) * y & \text{si } d < s \end{cases}$$

La ganancia esperada diaria $G(s, x, y)$ será la media de la función ganancia diaria respecto de la distribución $P(D = d)$, es decir

$$G(s, x, y) = \sum_{d} g(s, x, y, d) * P(D = d)$$

y por tanto nuestro problema consiste en

$$\text{Encontrar } s^*(x, y) = \arg \max_{s} G(s, x, y)$$

para unos valores dados de x e y.

Supongamos que la demanda diaria D se distribuye de una de las siguientes formas:

(a) $P(D = d)$ se distribuye uniformemente entre 0 y 99, es decir,

$$P(D = d) = \frac{1}{100}, \quad \forall d = 0, \ldots, 99$$

(b) $P(D = d)$ es proporcional a $100 - d$, $\forall d = 0, 1, 2, \ldots, 99$, es decir,

$$P(D = d) = \frac{100 - d}{5050}$$

(c) $P(D = d)$ tiene la siguiente distribución "triangular"

$$P(D = d) = \begin{cases} \frac{d}{2500} & \text{si } 0 \leq d < 50 \\ \\ \frac{100-d}{2500} & \text{si } 50 \leq d \leq 99 \end{cases}$$

Para resolver el problema de forma analítica, bastaría calcular la expresión funcional de $G(s, x, y)$ y encontrar el valor de s que maximiza esa expresión. Por ejemplo, haciendo los desarrollos apropiados, para el caso de la distribución de demanda uniforme (caso (a) anterior), la expresión que se obtiene es

$$G(s, x, y) = \frac{1}{200}\left(-(x + y)s^2 + (199x - y)s\right)$$

de donde el valor óptimo de s puede calcularse fácilmente como

$$s^* = \frac{199x - y}{2(x + y)}$$

Por ejemplo, si $x = 10$ e $y = 5$, el valor obtenido es $s^* = 66{,}17$. Como el valor buscado tiene que ser entero, teniendo en cuenta la forma de la función, tiene que ser $s^* = 66$ o $s^* = 67$. En este caso el verdadero óptimo es $s^* = 66$.

Tarea opcional: Comprobad (haciendo vuestros propios desarrollos) que la expresión de la ganancia esperada para la distribución uniforme es correcta. Intentad hacer lo mismo para las otras distribuciones de demanda[1] *(casos (b) y (c) anteriores).*

1.3. Modelización por Monte Carlo

Una forma alternativa de resolver (aproximadamente) este problema es construir un modelo de simulación del mismo. Para ello, la idea es simplemente estimar la ganancia esperada (por muestreo) en vez de calcularla de forma exacta: se utiliza un generador de datos que produzca datos a partir de la distribución de demanda diaria que se vaya a utilizar y se calcula la ganancia

[1] El empleo del software *Mathematica* puede seros de gran utilidad.

diaria para esa demanda. Entonces se repite este proceso muchas veces y al final se calcula el promedio de estas ganancias diarias. Es como si observásemos el sistema real durante muchos días, registrando cada día la ganancia obtenida, y estimásemos la ganancia esperada como el promedio de esas ganancias. Lo esencial de un modelo de simulación de Monte Carlo es *muestrear* el sistema en estudio para estimar las características de ese sistema a partir de los datos de la muestra.

En nuestro caso, y supuesto que ya disponemos del generador de datos para la distribución de la demanda, el núcleo del proceso sería el siguiente:

```
sum=0.0; sum2=0.0;
for (i=0; i<veces; i++)
 {
  demanda=genera_demanda();
  if (s>demanda) ganancia=demanda*x-(s-demanda)*y;
  else ganancia=s*x;
  sum+=ganancia;
  sum2+=ganancia*ganancia;
 }
ganancianesperada=sum/veces;
desviaciont=sqrt((sum2-veces*ganancianesperada*ganancianesperada)/(veces-1))
```

Con este procedimiento podemos calcular la ganancia esperada (y la desviación típica) para un valor concreto de s. Repitiendo este proceso para diferentes valores de s podemos calcular cuál de ellos produce la mayor ganancia esperada, y ese valor será la solución (aproximada) de nuestro problema.

Tarea: Construid un modelo de Monte Carlo, empleando en cada caso los generadores de datos para la distribución de la demanda diaria que después se especificarán, y experimentad con ellos para diferentes valores de x e y (por ejemplo $x = 10$, $y = 1$; $x = 10$, $y = 5$; $x = 10$, $y = 10$) y diferentes valores de `veces` *(por ejemplo* `veces`*=100, 1000, 5000, 10000, 100000,...). Contrastad, si es posible, los resultados obtenidos por Monte Carlo con los proporcionados por los métodos analíticos.*

1.3.1. Generadores de Datos

Para desarrollar completamente los modelos anteriores es necesario construir los generadores de datos correspondientes. Hasta que no se comente en clase de teoría el método de tablas de búsqueda para la construcción de generadores de datos discretos, se pueden utilizar los siguientes procedimientos/funciones (código en lenguaje C):

```
double uniforme() //Genera un número uniformemente distribuido en el
                  //intervalo [0,1) a partir de uno de los generadores
                  //disponibles en C. Lo utiliza el generador de demanda
{
 int t = random();
 double f = ((double)RAND_MAX+1.0);
```

```
 return (double)t/f;
}

double* construye_prop_a(int n) //Construye la tabla de búsqueda de
                                //tamaño n para la distribución de
                                //la demanda del apartado (a).
{
int i;
double* temp;
if ((temp = (double*) malloc(n*sizeof(double))) == NULL)
  {
   fputs("Error reservando memoria para generador uniforme\n",stderr);
   exit(1);
  }
temp[0] = 1.0/n;
for (i=1;i<n;i++)
  temp[i] = temp[i-1]+1.0/n;
return temp;
}

double* construye_prop_b(int n) //Construye la tabla de búsqueda de
                                //tamaño n para la distribución de
                                //la demanda del apartado (b).
{
int i, max;
double* temp;
if ((temp = (double*) malloc(n*sizeof(double))) == NULL)
  {
   fputs("Error reservando memoria para generador proporcional\n",stderr);
   exit(1);
  }
max = (n/2)*(n+1);
temp[0] = n*1.0/max;
for (i=1;i<n;i++)
  temp[i] = temp[i-1]+(double)(n-i)/max;
return temp;
}

double* construye_prop_c(int n) //Construye la tabla de búsqueda de
                                //tamaño n para la distribución de
                                //la demanda del apartado (c).
{
int i, max;
double* temp;
if ((temp = (double*) malloc(n*sizeof(double))) == NULL)
```

```
  {
   fputs("Error reservando memoria para generador triangular\n",stderr);
   exit(1);
  }
max = n*n/4;
temp[0] = 0.0;
for (i=1;i<(n/2);i++)
  temp[i] = temp[i-1]+(double)i/max;
for (i=(n/2);i<n;i++)
  tem[i] = temp[i-1]+(double)(n-i)/max;
return temp;
}

int genera_demanda(double* tabla,int tama) // Genera un valor de la
        // distribución de la demanda codificada en tabla, por el
        // método de tablas de búsqueda.
        // tama es el tamaño de la tabla, 100 en nuestro caso particular
{
int i;
double u = uniforme();
i = 0;
while((i<tama) && (u>=tabla[i]))
  i++;
return i;
}
```

El programa principal debe también incluir lo siguiente:

```
srand(time(NULL)); //Inicializa el generador de números pseudoaleatorios
tablabdemanda = construye_prop_?(100); //Construye la tabla de búsqueda
```

donde el carácter "`?`" se reemplaza por el valor "`a`", "`b`" o "`c`", según el generador que se vaya a usar. Las llamadas al generador de datos en el bucle del programa, que antes hemos indicado como `genera_demanda()`, realmente deben hacerse mediante[2]

```
demanda = genera_demanda(tablabdemanda,100) //Cada vez que se necesite un
                                    //valor del generador de demanda
```

[2]Obsérvese que los procedimientos están parametrizados, de forma que los generadores para las distribuciones de probabilidad especificadas se obtienen para el valor `tama = n = 100`, pero cambiando este parámetro por cualquier número entero obtenemos los generadores para distribuciones del mismo tipo, es decir uniforme entre 0 y $n-1$ en el caso (a), proporcional a $n-d$ en el caso (b) y "triangular" entre 0 y $n-1$ con vértice en $n/2$ en el caso (c).

1.4. Modificaciones del Modelo

Imaginad ahora que las circunstancias fuesen distintas, de forma que x sigue siendo la ganancia obtenida por unidad vendida, pero que el establecimiento puede devolver las unidades no vendidas (de modo que no hay pérdidas por unidad no vendida, $y = 0$), aunque sí hay que pagar una cantidad fija de z euros en concepto de gastos de devolución de las unidades no vendidas (dicha cantidad es independiente de cuántas unidades queden sin vender, salvo que se vendan todas, en cuyo caso no hay gasto ninguno).

Con este planteamiento, la función ganancia diaria sería ahora:

$$g(s,x,z,d) = \begin{cases} x * s & \text{si } d \geq s \\ \\ x * d - z & \text{si } d < s \end{cases}$$

Tarea: Modificad el modelo de Monte Carlo y repetid la experimentación, contrastando los resultados con los anteriores. Opcionalmente intentad calcular las nuevas expresiones analíticas de la ganancia esperada para las distribuciones de demanda.

Por último, suponed la siguiente situación: si el valor z es relativamente grande, tal vez no le interese al establecimiento pagar esa cantidad siempre que quede alguna unidad sin vender. Cuando el número de unidades no vendidas sea pequeño, quizás le compense asumir una pérdida de y euros por unidad no vendida y no pagar los z euros de gastos de devolución. Más precisamente, esto ocurrirá cuando la cantidad $(s - d) * y$ (la pérdida total por unidades no vendidas) sea menor que z. Con este planteamiento, la nueva función de ganancia diaria es:

$$g(s,x,y,z,d) = \begin{cases} x * s & \text{si } d \geq s \\ \\ x * d - \text{mín}\{z,\, (s-d) * y\} & \text{si } d < s \end{cases}$$

Tarea: repetid todo lo anterior para esta nueva situación, y comparad los resultados.

Capítulo 2

Generadores de Datos

2.1. Mejorando los Generadores

Los generadores de datos empleados en el capítulo anterior se han construido aplicando directamente el método de tablas de búsqueda de forma "ingenua", pero se pueden refinar para que sean más eficientes:

1. Simplemente reordenando los valores de la variable en orden decreciente de probabilidad *antes* de construir la tabla, se consiguen generadores más eficientes (en promedio).

   Construid estos generadores reordenados y comparad su eficiencia con los originales[1].

2. También se puede utilizar una técnica de búsqueda más eficiente (por ejemplo búsqueda binaria) para buscar en la tabla.

   Implementad los tres generadores con un método de búsqueda más eficiente que una simple búsqueda lineal, y comparadlos con los anteriores.

3. Para el generador del caso (a), es posible mejorar más aún la eficiencia del generador, de forma que su tiempo de ejecución sea constante (evita cualquier proceso explícito de búsqueda).

   Deducid el método, implementadlo y comparadlo con los anteriores.

Haced las comparaciones entre las distintas implementaciones de cada generador de forma independiente del modelo de Monte Carlo, mediante un programa que simplemente haga multitud de llamadas a cada generador y calcule el tiempo de ejecución empleado por cada uno.

2.2. ¡Cuidado con las implementaciones de los generadores básicos!

Implementad los generadores congruenciales lineales

$$x_{n+1} = (2061x_n + 4321) \bmod m$$

[1]Para el caso (b) el generador anterior ya tenía los valores ordenados en orden decreciente, por lo que en este caso no tenemos nada nuevo. En el caso (a) tampoco sirve, porque también los valores están "ordenados" de forma decreciente (puesto que todos son iguales)

y

$$x_{n+1} = (2060x_n + 4321) \bmod m$$

con $m = 10^4$, *utilizando aritmética entera y aritmética real de diferentes formas (que se detallan a continuación). ¿Qué periodos se obtienen en cada caso? ¿Cuál o cuáles son las implementaciones correctas? ¿Por qué?*

Para calcular el valor $y = (a * x + c) \bmod m$, emplead los siguientes métodos (utilizando lenguaje C):

- Aritmética entera: Usad directamente la instrucción que calcula el resto módulo m:

  `x=(a*x+c)%m`, donde `x` es una variable entera (usad enteros largos para evitar problemas)

- Aritmética real "artesanal": Se calcula el valor dividiendo $a * x + c$ entre m, restándole su parte entera y multiplicando por m. En este caso `x` es una variable float. Probad también declarándola de tipo double:

  ```
  x=(a*x+c)/m
  ```

  ```
  x=(x-(int)x)*m
  ```

- Aritmética real "artesanal" corregida: Se calcula el valor igual que antes, pero intentando corregir los errores de redondeo:

  ```
  x=(a*x+c)/m
  ```

  ```
  x=(x-(int)x)*m
  ```

  ```
  x=(int)(x+0.5)
  ```

- Aritmética real usando `fmod`: Esta función $fmod(u, v)$ devuelve el resto de dividir u entre v, es decir, devuelve $u - n * v$ donde n es el cociente de u/v redondeado (por defecto) a un entero:

  `x=fmod((a*x+c),m)`, donde `x` es una variable double

Para calcular el periodo del generador, hay que ir almacenando los valores generados en un array, y cada vez que se genere un valor nuevo, se compara con todos los anteriores. Tened cuidado en caso de que los números a comparar sean float o double.