

6.- Generadores de números aleatorios

6.1- Generadores de números aleatorios U(0,1)

6.1.1- Generadores lineales congruentes

La generación de números aleatorios consiste en obtener un posible valor numérico de una variable aleatoria, de acuerdo con su distribución de probabilidad. En este apartado, describiremos algunos métodos de generación de números aleatorios de distribución uniforme en el intervalo $[0,1]$, U(0,1).

La importancia fundamental de esta distribución radica en el hecho de que, transformando las variables aleatorias IID U(0,1) de forma adecuada, pueden obtenerse variables aleatorias para cualquiera de las demás distribuciones (normal, gamma, etc.) y algunos procesos aleatorios, por ejemplo procesos de Poisson no estacionarios. Estas transformaciones serán descritas en el siguiente apartado.

Gran parte de los generadores de números aleatorios empleados en la actualidad son *generadores lineales congruentes*, LCGs (linear congruential generators), introducidos por Lehmer en 1951. Se define una secuencia de enteros Z_1, Z_2, \dots mediante la forma recursiva:

$$Z_i = (aZ_{i-1} + c) \pmod{m}$$

donde m (el *módulo*), a (el *multiplicador*), c (el *incremento*) y Z_0 (el *valor inicial*) son enteros no negativos. Como vemos, para obtener Z_i , dividimos $aZ_{i-1} + c$ por m y el resto de la división es Z_i . Así pues, $0 \leq Z_i \leq m-1$, y para obtener los deseados números aleatorios U_i (para $i:1,2,\dots$) sobre $[0,1]$, hacemos $U_i = \frac{Z_i}{m}$. Además de no negativos, los enteros m , a , c y Z_0 deben satisfacer $0 < m$, $a < m$, $c < m$ y $Z_0 < m$.

Inmediatamente pueden hacerse dos objeciones a los LCGs. La primera objeción es que los Z_i 's definidos de esta forma no son en absoluto aleatorios: puede demostrarse por inducción que:

$$Z_i = \left[a^i Z_0 + \frac{c(a^i - 1)}{a - 1} \right] \pmod{m}$$

de modo que cada Z_i está completamente determinado por m , a , c y Z_0 . Sin embargo, escogiendo cuidadosamente estos cuatro parámetros, puede conseguirse que los Z_i 's hagan que los correspondientes U_i 's se comporten como IID U(0,1) cuando se les somete a una variedad de test.

La segunda objeción es que los U_i 's sólo pueden tomar los valores racionales $0, \frac{1}{m}, \frac{2}{m}, \dots, \frac{m-1}{m}$, con lo cual la probabilidad de obtener un valor entre, por ejemplo, $\frac{0.1}{m}$ y $\frac{0.9}{m}$, es cero, mientras que debiera de ser igual a $\frac{0.8}{m} > 0$. Como veremos, el módulo m suele escogerse muy grande, 10^9 o superior, de modo que los valores que pueden tomar los U_i 's en el intervalo $[0,1]$ sean muy densos.

Ejemplo. Definamos un LCG con $m=16$, $a=5$, $c=3$ y $Z_0=7$. En la siguiente tabla se dan los valores de Z_i y U_i (con 3 cifras decimales) para $i:1,2,\dots,33$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Z_i	7	6	1	8	11	10	5	12	15	14	9	0	3	2	13	4	7
U_i	-	0.375	0.063	0.500	0.688	0.625	0.313	0.750	0.938	0.875	0.563	0.000	0.188	0.125	0.813	0.250	0.438

i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Z_i	6	1	8	11	10	5	12	15	14	9	0	3	2	13	4	7	6
U_i	0.375	0.063	0.500	0.688	0.625	0.313	0.750	0.938	0.875	0.563	0.000	0.188	0.125	0.813	0.250	0.438	0.375

El LCG $Z_i = (5Z_{i-1} + 3) \pmod{16}$ con $Z_0 = 7$

obsérvese que $Z_{33} = Z_{17} = Z_1 = 6$, $Z_{18} = Z_2 = 1$, y así sucesivamente. Es decir, desde $i=17$ hasta 32 obtenemos los mismos valores de Z_i y U_i que desde $i=1$ hasta 16.

Este comportamiento cíclico del LCG es inevitable, ya que, por definición, si toma un valor que ya a aparecido previamente, genera entonces exactamente la misma secuencia de valores y este ciclo se repite sin fin. La longitud del ciclo se llama *periodo del generador* y lo notaremos p . Para LCGs, Z_i depende sólo de su valor anterior, Z_{i-1} , y como $0 \leq Z_i \leq m-1$, se cumple $p \leq m$. Cuando $p=m$ se dice que el generador lineal congruente tiene *periodo completo* (es el caso de LCG del ejemplo).

Dado que los proyectos de simulación de gran escala usan cientos de miles de números aleatorios, es deseable tener LCGs de grandes periodos. También interesa tener LCGs de grandes periodos, de modo que tengamos la seguridad de que cada entero entre 0 y $m-1$ sólo aparezca una vez en cada ciclo, lo cual contribuirá a la uniformidad de los U_i 's. Pese a esto, incluso los LCGs de periodo completo pueden presentar comportamiento no uniforme en segmentos dentro de ciclo. Por ejemplo, si generamos sólo $m/2$ Z_i 's consecutivos, puede quedar grandes "huecos" en la secuencia $0,1,2,\dots,m-1$ de posibles valores.

El LCG definido $Z_i = (aZ_{i-1} + c) \pmod{m}$ tiene periodo completo si y sólo si se satisfacen las siguientes tres condiciones:

- (a).- El único entero positivo que divide exactamente tanto m como c es el uno.
- (b).- Si q es un número primo (divisible sólo por si mismo y por el uno) que divide m , entonces q divide a $(a-1)$.
- (c).- Si m es divisible por 4, entonces $(a-1)$ también es divisible por 4.

Discutamos primero la elección de m , de modo que sean de periodo completo ($p=m$) y largo (m grande). La división por m necesaria para obtener la secuencia de Z_i 's es una operación aritmética relativamente lenta y que sería conveniente evitar realizar explícitamente. Una elección del módulo m acertada en todos los aspectos es $m = 2^b$ (por ejemplo, si $b=31$, entonces $m \approx 2.15E + 9$), donde b es el número de bits de la palabra binaria en que almacena el dato en memoria el ordenador. Si almacena los enteros en palabras de 32 bits y el más significativo se reserva para el signo, entonces $b=31$. La elección $m = 2^b$ permite evitar realizar explícitamente la división por m en la mayoría de los ordenadores haciendo uso del rebosamiento entero (integer overflow). El mayor entero que puede representarse en una palabra binaria de 31 bits es $2^b - 1$, con lo cual, si se intenta almacenar un entero W mayor, con h ($h > b$) bits, se producirá la pérdida de los $h-b$ bits más significativos. Lo que queda (los b bits menos significativos de W), es precisamente $W(\text{mod } 2^b)$.

Ejemplo. Supongamos que el LCG del ejemplo anterior se programa en un ordenador "prehistórico" de $b=4$ bits por palabra para el almacenamiento de datos. Se escoge entonces $m=16=2^b$. ¿Como puede utilizarse el rebosamiento para obtener $Z_7 = 12$ a partir de $Z_6 = 5$? La representación binaria de $5Z_6 + 3 = 28$ es la palabra 11100. Como nuestro ordenador de 4 bits sólo puede almacenar palabras de 4 bits, el bit más significativo se pierde, quedando almacenado 1100, que es la representación binaria de $Z_7 = 12$.

Los LCGs se clasifican dependiendo de si $c > 0$ (LCGs mixtos) o si, por el contrario, $c=0$ (LCGs multiplicativos).

6.1.2- Generadores lineales congruentes mixtos

Los LCGs mixtos ($c > 0$) no suelen emplearse, ya que los multiplicativos son más sencillos y, en general, presentan similares prestaciones. Dos de los LCGs mixtos más ampliamente usados son, para $b=35$:

$$Z_i = (5^{15} Z_{i-1} + 1) (\text{mod } 2^{35}) \quad (\text{Coveyou y MacPherson})$$

y, para $b=31$:

$$Z_i = (314,159,269 Z_{i-1} + 453,806,245) (\text{mod } 2^{31}) \quad (\text{Kobayashi})$$

6.1.3- Generadores lineales congruentes multiplicativos

Los *LCGs multiplicativos* ($c=0$) son preferibles en cuanto a su sencillez, ya que no es necesario determinar c , sin embargo, no satisfacen la condición (a) de periodo completo, ya que, por ejemplo, m es positivo y divide tanto a m como a $c=0$. Sin embargo, si se escogen m y a cuidadosamente, es posible obtener periodo $p=m-1$.

Para los LCGs multiplicativos, si bien la elección $m = 2^b$ es ventajosa ya que evita la división explícita, por contra, da lugar a periodos $p \leq 2^{b-2}$, a lo máximo un cuarto del completo, es decir, sólo un cuarto de los enteros entre 0 y $m-1$ pueden obtenerse como valores para los Z_i 's. En efecto, $p = 2^{b-2}$ si Z_0 es impar y a es de la forma $8k+3$ ó $8k+5$ para algún $k=0,1,2,\dots$. Generalmente no puede predecirse donde caerán estos $m/4$ enteros, con lo cual pueden producirse "agujeros" inaceptablemente grandes en los Z_i 's generados. Estos dos problemas han inducido a la búsqueda de otras formas de especificar m .

En lugar de escoger $m = 2^b$, se ha propuesto (Hutchinson) emplear el mayor número primo menor que 2^b para los generadores LCGs multiplicativos. Si m es primo y a es un *elemento primitivo módulo m* , es decir, es tal que el entero más pequeño, i , para el cual $a^i - 1$ es divisible por m es $i=m-1$, entonces, el periodo es $p=m-1$, obtenemos los enteros $1,2,3,\dots,m-1$ exactamente una vez en cada ciclo, de modo que Z_0 puede ser cualquier entero de 1 a $m-1$. Los generadores de este tipo se llaman *generadores LCGs multiplicativos de módulo primo* (PMMLCGs).

Inmediatamente surge una pregunta relativa a los PMMLCGs: cómo obtener a (un elemento primo módulo m), y un inconveniente: dado que el módulo no es $m = 2^b$ no puede usarse el desbordamiento entero para realizar la división módulo m . Una técnica para evitar la división explícita en este caso, y que también emplea el desbordamiento, es la *división simulada*. En los PMMLCGs, m es de la forma $2^b - q$, donde q es algún entero positivo. Para obtener $Z_i = (aZ_{i-1}) \pmod{2^b - q}$, a partir de Z_{i-1} , definimos $Z_i^* = (aZ_{i-1}) \pmod{2^b}$, que puede calcularse por rebosamiento. Si k es el mayor entero que es menor o igual que $\frac{aZ_{i-1}}{2^b}$, entonces:

$$Z_i = \begin{cases} Z_i^* + kq & \text{si } Z_i^* + kq < 2^b - q \\ Z_i^* + kq - (2^b - q) & \text{si } Z_i^* + kq \geq 2^b - q \end{cases}$$

Ejemplos de PMMLCGs con buenas propiedades son:

Para $b=35$, el mayor primo menor que 2^{35} es $m = 2^{35} - 31 = 34,359,738,337$ y una elección acertada de a puede ser $a = 5^5 = 3125$ (que es un elemento primitivo módulo $2^{35} - 31$).

Para $b=31$, el mayor primo menor que 2^{31} es $m = 2^{31} - 1 = 2,147,483,647$ y dos posibles buenas elecciones de a son $a_1 = 7^5 = 16807$ y $a_2 = 630,360,016$.

6.1.4- Otros tipos de generadores

Aunque los generadores lineales congruentes son los más ampliamente usados, existen otros tipos de generadores desarrollados para obtener periodos más largos y mejores propiedades. Sin embargo, a menudo, un LCG con parámetros adecuados puede tener características tan buenas (y a veces mejores) que estas alternativas más complicadas.

Los LCGs pueden considerarse un caso especial de los generadores definidos por:

$$Z_i = g(Z_{i-1}, Z_{i-2}, \dots) \pmod{m}$$

donde $g(\)$ es una función determinista dada. Los Z_i 's definidos de esta manera, están comprendidos entre 0 y $m-1$, y los números aleatorios $U(0,1)$ vienen dados por $U_i = \frac{Z_i}{m}$. Por ejemplo:

$g(Z_{i-1}, Z_{i-2}, \dots) = a'Z_{i-1}^2 + aZ_{i-1} + c$ produce un *generador congruente cuadrático*.

$g(Z_{i-1}, Z_{i-2}, \dots) = a_{i-1}Z_{i-1} + a_2Z_{i-2} + \dots + a_qZ_q$, en particular, el generador de Fibonacci (no recomendable) $g(Z_{i-1}, Z_{i-2}) = Z_{i-1} + Z_{i-2}$.

Se han realizado investigaciones encaminadas a la generación de números aleatorios combinando los obtenidos a partir de dos generadores (normalmente LCG) diferentes. Se los llama *generadores compuestos*.

Los generadores de Tausworthe, relacionados con métodos criptográficos, operan directamente con bits para obtener números aleatorios. Presentan la ventaja frente a los LCGs de ser esencialmente independientes de la longitud de palabra en el ordenador, y de permitir periodos muy grandes (tales como $2^{521} - 1 > 10^{156}$) incluso con ordenadores de 16 bits. Se define una secuencia de dígitos binarios b_1, b_2, \dots mediante la relación de recurrencia:

$$b_i = (c_1b_{i-1} + c_2b_{i-2} + \dots + c_qb_{i-q}) \pmod{2}$$

donde c_1, c_2, \dots, c_q son constantes de valor cero o uno, de modo que sólo dos de ellos son distintos de cero, con lo cual, queda:

$$b_i = (b_{i-r} + b_{i-q}) \pmod{2}$$

donde r y q satisfacen $0 < r < q$. La ejecución de esta relación de recurrencia se acelera si se tiene en cuenta que es equivalente a:

$$b_i = \begin{cases} 0 & \text{si } b_{i-r} = b_{i-q} \\ 1 & \text{si } b_{i-r} \neq b_{i-q} \end{cases}$$

Para inicializar la secuencia $\{b_i\}$ es necesario especificar los q primeros b_i 's, que es equivalente a la especificación de Z_0 en los LCGs.

Ejemplo. Sean $r=3$, $q=5$ y $b_1 = b_2 = b_3 = b_4 = b_5 = 1$. Entonces, para $i \geq 6$, b_i es el "or exclusivo" de b_{i-3} con b_{i-5} . Los primeros 42 b_i 's son:

1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0

obsérvese que el periodo de los bits es $2^q - 1 = 31$.

Existen diferentes métodos de transformar la secuencia $\{b_i\}$ en números aleatorios $U(0,1)$. Un método consiste en agrupar las cadenas de i bits consecutivos para formar el entero binario de i bits entre 0 y $2^i - 1$, que es entonces dividido por 2^i . En la secuencia del ejemplo anterior, escogiendo $i=4$, resulta la siguiente secuencia de números aleatorios: 15/16, 8/16, 13/16, 13/16, 4/16, 2/16, 5/16, 9/16, 15/16, 1/16,... Sin embargo, i no debe ser superior al tamaño de palabra en el ordenador. Otra posibilidad es usar los i primeros bits consecutivos para el primer entero, saltar algún número de los siguientes bits, usar los siguientes i bits para el siguiente entero, etc.

6.1.5- Ejercicio.

El alumno deberá programar, en un lenguaje de programación de su elección, los siguientes generadores de números aleatorios $U(0,1)$:

- Un LCG mixto, no realizando la división explícitamente, sino haciendo uso del rebosamiento entero.
- Un PMMLCG con división simulada.
- Un generador de Tausworthe, con parámetros de entrada r, q, b_1, \dots, b_q, i . Para transformar la secuencia de bits en números aleatorios, agrúpense las cadenas de i bits consecutivos. Ejecutar el generador para el caso particular $r = 4, q = 5, b_1 = \dots = b_q = 1$, mostrando los 64 primeros bits de la secuencia y números aleatorios obtenidos escogiendo $i=4$.

Deberán entregarse una memoria escrita con los resultados y un disquete con los fuentes.

6.2- Test empíricos de generadores de números aleatorios U(0,1)

Como hemos visto, los generadores de números aleatorios empleados en simulación son completamente deterministas, con lo cual, sólo cabe esperar que los U_i 's generados "parezcan" variables U(0,1) IID. En esta sección describiremos varios test para cuantificar este "parecido".

Algunos programas comerciales traen incorporado, como parte del software disponible, un generador de números aleatorios. Es muy recomendable identificar qué tipo de generador es, cuales son sus parámetros numéricos y someterlo al menos a algún tipo de test empírico antes de usarlo en simulación. Existen dos tipos diferenciados de test: los empíricos y los teóricos.

6.2.1- Test de los momentos

Para n números aleatorios U_i 's distribuidos U(0,1), el primer, segundo y tercer momento deben valer, respectivamente $\frac{1}{2}$, $\frac{1}{3}$ y $\frac{1}{4}$. Esto es:

$$\frac{1}{n} \sum_{i=1}^n U_i = \frac{1}{2} \quad ; \quad \frac{1}{n} \sum_{i=1}^n U_i^2 = \frac{1}{3} \quad ; \quad \frac{1}{n} \sum_{i=1}^n U_i^3 = \frac{1}{4}$$

cuando n es muy grande.

6.2.2- Test de frecuencia usando el método chi-cuadrado

Los *test empíricos* se basan en determinar estadísticamente el parecido entre los U_i 's generados y las variables aleatorias U(0,1) IID. Veamos un test empírico (caso particular del *test chi-cuadrado*) diseñado para comprobar si los U_i 's están uniformemente distribuidos en $[0,1]$.

Se divide $[0,1]$ en k subintervalos de la misma longitud y se generan U_1, U_2, \dots, U_n . Como regla, suele aceptarse escoger k mayor que 100 y n al menos 5 veces mayor que k . Para $j: 1, 2, \dots, k$, se define f_j como la frecuencia observada en el intervalo j -ésimo: el número de los U_i 's que caen en el subintervalo j -ésimo. La frecuencia teórica en el intervalo j -ésimo es $\frac{n}{k}$. Para n grande,

$$\chi^2 = \frac{k}{n} \sum_{j=1}^k \left(f_j - \frac{n}{k} \right)^2 = \frac{\sum (\text{observado} - \text{esperado})^2}{\text{esperado}}$$

tiene aproximadamente una distribución chi-cuadrado, con $k-1$ grados de libertad, si se satisface la hipótesis H_0 : los U_i 's son variables aleatorias U(0,1) IID. Un valor pequeño de χ^2 indica una buena coincidencia entre las frecuencias teóricas y las experimentales.

Así pues, rechazaremos la hipótesis, con nivel de significación α , si $\chi^2 > \chi_{k-1,1-\alpha}^2$, donde $\chi_{k-1,1-\alpha}^2$ es el punto crítico $1-\alpha$ de la distribución chi-cuadrado con $k-1$ grados de libertad. Para valores de k grandes, como los empleados aquí, puede usarse la aproximación:

$$\chi_{k-1,1-\alpha}^2 \approx (k-1) \left\{ 1 - \frac{2}{9(k-1)} + z_{1-\alpha} \left[\frac{2}{9(k-1)} \right]^{\frac{1}{2}} \right\}^3$$

donde $z_{1-\alpha}$ es el punto crítico $1-\alpha$ de la distribución $N(0,1)$.

Ejemplo. La salida de un generador de números aleatorios de un solo dígito es:

Dígito	0	1	2	3	4	5	6	7	8	9
f_j	30	20	35	36	17	14	29	20	18	31

En este caso, $n = 250$, $k = 10$, $\frac{n}{k} = 25$.

$$\chi^2 = \frac{1}{25} \sum_{i=1}^{10} (f_i - 25)^2 = 23.29$$

Para 9 grados de libertad y un nivel de significación del 5% ($\alpha = 0.05$) tenemos, de la tabla IV, el punto crítico $\chi_{9,0.95}^2 = 16.919$. Como $\chi^2 = 23.29 > \chi_{9,0.95}^2 = 16.919$, generador de números aleatorios de un solo dígito no se comportan al nivel 5% como un generador de variables aleatorias $U(0,1)$ IID.

Ejemplo. Aplicamos el test chi-cuadrado al PMMLCG: $Z_i = (7^5 Z_{i-1}) \pmod{2^{31} - 1}$.

Como valor inicial tomamos $Z_0 = 12,345,678$. Para examinar los 12 bits más significativos de los U_i 's, hacemos $k = 2^{12} = 4096$ y escogemos $n = 2^{16} = 65536$. Obtenemos $\chi^2 = 4001.625$ y, de la aproximación anterior para el punto crítico, $\chi_{4095,0.90}^2 \approx 4211.402$, con lo cual no rechazamos la hipótesis al nivel 0.1. Así pues, estos U_i 's en particular producidos por el generador no se comportan de manera *significativamente diferente* (al nivel 0.1, al menos) de lo esperado para variables aleatorias $U(0,1)$ IID.

6.2.3- Test de las carreras

Un test útil para comprobar independencia es el *test de las carreras* (no comprueba uniformidad). Examinamos la secuencia U_i (o, equivalentemente, la secuencia Z_i) buscando secuencias de U_i 's monótonas crecientes de la máxima longitud posible. A cada una de estas secuencias se la llama *carrera*.

Ejemplo. Consideremos la secuencia U_1, \dots, U_{10} :

0.855, 0.108, 0.226, 0.032, 0.132, 0.055, 0.545, 0.642, 0.870, 0.104

en la cual, hay las siguientes carreras:

0.855
0.108, 0.226
0.032, 0.132
0.055, 0.545, 0.642, 0.870
0.104

En una secuencia de n U_i 's se cuenta el número de carreras de longitud 1, 2, 3, 4, 5, y ≥ 6 , definiéndose:

$$r_i = \begin{cases} \text{numero de carreras de longitud } i & \text{para } i = 1, 2, 3, 4, 5 \\ \text{numero de carreras de longitud } \geq 6 & \text{para } i = 6 \end{cases}$$

En la secuencia anterior: $r_1 = 2$, $r_2 = 2$, $r_3 = 0$, $r_4 = 1$, $r_5 = 0$ y $r_6 = 0$.

El test estadístico es:

$$R = \frac{1}{n} \sum_{i=1}^6 \sum_{j=1}^6 a_{ij} (r_i - nb_i)(r_j - nb_j) = \frac{1}{n} \vec{v}^t A \vec{v}$$

donde a_{ij} es el elemento (i, j) de la matriz:

$$A = \begin{pmatrix} 4529.4 & 9044.9 & 13568 & 18091 & 22615 & 27892 \\ 9044.9 & 18097 & 27139 & 36187 & 45234 & 55789 \\ 13568 & 27139 & 40721 & 54281 & 67852 & 83685 \\ 18091 & 36187 & 54281 & 72414 & 90470 & 111580 \\ 22615 & 45234 & 67852 & 90470 & 113262 & 139476 \\ 27892 & 55789 & 83685 & 111580 & 139476 & 172860 \end{pmatrix}$$

b_i viene dado por:

$$\vec{b}^t = (b_1, b_2, b_3, b_4, b_5, b_6) = \left(\frac{1}{6}, \frac{5}{24}, \frac{11}{120}, \frac{19}{720}, \frac{29}{5040}, \frac{1}{840} \right)$$

y se define el vector columna \vec{V} de la forma: $\vec{V} = \vec{r} - n\vec{b}$

Para n grande (se recomienda $n \geq 4000$), R tiene aproximadamente una distribución chi-cuadrado con 6 grados de libertad bajo la hipótesis de que las U_i 's son variables aleatorias IID.

Ejemplo. Supongamos que sometemos un generador al test de las carreras con $n=65536$. Obtenemos:

$$\vec{r}' = (r_1, r_2, r_3, r_4, r_5, r_6) = (10864, 13695, 5884, 1774, 401, 84)$$

de donde se obtiene $R=8.206$. Dado que $\chi^2_{6,0.90} = 10.6$, *no rechazamos* (al nivel 0.1) la hipótesis de que los U_i 's son variables aleatorias IID.

El test de las carreras suele ser más potente que el chi-cuadrado: algunos generadores pasan el test chi-cuadrado, pero no el de las carreras. Es recomendable pasar ambos test al generador, primero el de las carreras y, si lo supera, el chi-cuadrado (y todos aquellos otros test, de los innumerables existentes, de que se disponga).

Una desventaja de los test empíricos es que son locales, es decir, sólo se examina el segmento del ciclo que se ha usado para generar los U_i 's en cuestión, no pudiéndose afirmar nada acerca del resto del ciclo. Esto puede considerarse, en cambio, una ventaja, ya que permite examinar sólo los números aleatorios que serán usados posteriormente en la simulación.

6.2.4- Ejercicio

Aplicar el test de las carreras y el test chi-cuadrado a los tres generadores programados en el ejercicio anterior. En el caso del generador de Tausworthe, emplear $r=3$, $q=31$.

Realizar una memoria escrita describiendo el procedimiento seguido y los resultados obtenidos.

6.3- Generación de variables aleatorias

6.3.1- Selección del algoritmo

Consideremos el problema de generar variables aleatorias con una distribución determinada conocida. Como veremos, el "ingrediente básico" para la generación de cualquier distribución o proceso aleatorio, siguiendo la mayoría de los métodos existentes, es una fuente de variables aleatorias $U(0,1)$ IID.

Factores a considerar en la selección del algoritmo para la generación de variables aleatorias dada su distribución son:

exactitud: que el algoritmo proporcione variables aleatorias con exactamente la distribución deseada.

eficiencia: capacidad de almacenamiento necesaria y velocidad de ejecución marginal (tiempo para generar cada variable aleatoria) y de arranque (tiempo necesario para iniciar la computación, empleado en determinar parámetros, construir tablas, etc.).

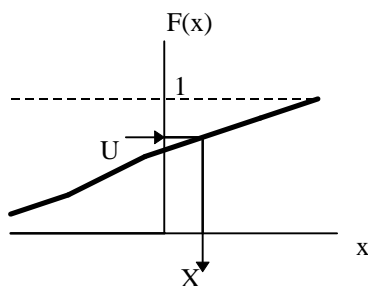
complejidad: conceptual y de programación.

robustez: el algoritmo debe ser igualmente eficiente para cualquier valor de los parámetros de la distribución.

6.3.2- Transformación inversa

Antes de describir los algoritmos específicos para cada distribución, examinemos las bases teóricas sobre las que descansan la mayoría de estos algoritmos.

Supongamos que queremos generar una variable aleatoria X continua con distribución de probabilidad acumulada $F(x)$ continua y estrictamente creciente cuando $0 < F(x) < 1$, es decir, si $x_1 < x_2$ y $0 < F(x_1) \leq F(x_2) < 1$, entonces $F(x_1) < F(x_2)$. Notemos F^{-1} la inversa de la función F . Entonces, un algoritmo para generar una variable aleatoria X con distribución F , llamado *método general de la transformación inversa*, es:



- Generar una variable aleatoria $U(0,1)$, U .

- Asignar $X = F^{-1}(U)$

Obsérvese que $F^{-1}(U)$ estará definido siempre, ya que $0 \leq U \leq 1$ y el rango de F es $[0,1]$, y además de forma única, ya que F es monótona creciente. La demostración del método, es decir, de que X esta distribuido F, $P\{X \leq x\} = F(x)$, es:

$$P\{X \leq x\} = P\{F^{-1}(U) \leq x\}$$

como F es una función monótona creciente de x, la desigualdad $a \leq b$ es equivalente a $F(a) \leq F(b)$:

$$P\{F^{-1}(U) \leq x\} = P\{F(F^{-1}(U)) \leq F(x)\} = P\{U \leq F(x)\} = F(x)$$

donde la última igual se verifica dado que U esta distribuida $U(0,1)$ y $0 \leq F(x) \leq 1$.

Ejemplo. Se desea generar X con distribución exponencial. La distribución F es:

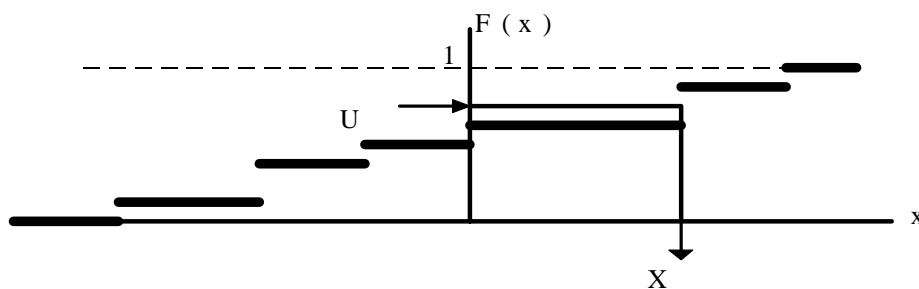
$$F(x) = \begin{cases} 1 - e^{-\frac{x}{\beta}} & \text{si } x \geq 0 \\ 0 & \text{en cualquier otro caso} \end{cases}$$

para calcular F^{-1} , resolvemos x de $u = F(x)$: $F^{-1}(u) = -\beta \ln(1-u)$. Así pues, para generar la variable X deseada, primero generamos U, distribuida $U(0,1)$, y calculamos $X = -\beta \ln(U)$. Usamos U en vez de 1-U porque, U y 1-U tienen la misma distribución $U(0,1)$ y de esta forma se evita tener que realizar la resta.

El método de la transformación inversa también es aplicable cuando X es discreta. En este caso, la distribución de probabilidad acumulada es:

$$F(x) = P\{X \leq x\} = \sum_{\{i: x_i \leq x\}} p(x_i)$$

donde, $p(x_i)$ es la probabilidad, $p(x_i) = P\{X = x_i\}$. Suponiendo que X sólo puede tomar los valores x_1, x_2, \dots y que $x_1 < x_2 < \dots$, el algoritmo es:



(1)- Generar una variable aleatoria $U(0,1)$, U.

(2)- Determinar el menor entero positivo I tal que $U \leq F(x_I)$, es decir,

$$\sum_{j=0}^{I-1} p(j) \leq U < \sum_{j=0}^I p(j) \text{ Asignar } X = x_I.$$

Para comprobar que el método de la transformación inversa discreta es válido, debe demostrarse que $P\{X = x_i\} = p(x_i)$ para todo i . Para $i=1$, tenemos $X = x_1$ si y sólo si $U \leq F(x_1) = p(x_1)$, ya que los x_i 's están ordenados en orden creciente. Como U esta distribuida $U(0,1)$, $P\{X = x_1\} = p(x_1)$, como se desea. Para $i \geq 2$, el algoritmo establece $X = x_i$, si y sólo si, $F(x_{i-1}) < U \leq F(x_i)$, ya que i es el menor entero positivo tal que $U \leq F(x_i)$.

$$P\{X = x_i\} = P\{F(x_{i-1}) < U \leq F(x_i)\} = F(x_i) - F(x_{i-1}) = p(x_i)$$

Ejemplo. En el ejemplo del inventario la cantidad de producto demandada era una variable aleatoria X discreta, cuyos valores podían ser 1,2,3 ó 4, con probabilidades 1/6, 1/3, 1/3, 1/6. Para generar X primero obtenemos U , distribuida $U(0,1)$, y escogemos X dependiendo de que parte del intervalo $[0,1]$ caiga U . Si $U \leq \frac{1}{6}$, entonces $X=1$. Si $\frac{1}{6} < U \leq \frac{1}{2}$, entonces $X=2$. Si $\frac{1}{2} < U \leq \frac{5}{6}$, entonces $X=3$. Si $\frac{5}{6} < U$, entonces $X=4$.

El método de la transformación inversa descrito en el ejemplo es muy intuitivo: se divide el intervalo $[0,1]$ en subintervalos contiguos de longitud $p(x_1), p(x_2), \dots$ y se asigna X dependiendo de cual de estos subintervalos contiene a U : si U cae en el subintervalo i -ésimo, de probabilidad $p(x_i)$, se asigna $X = x_i$.

Una posible dificultad en la aplicación del método en el caso continuo puede ser la imposibilidad de encontrar una expresión analítica para F^{-1} , como de hecho ocurre, por ejemplo, con la distribución normal y gamma, debiéndose utilizar métodos numéricos.

El método es fácilmente aplicable cuando la distribución esta truncada. Supongamos que f es una función densidad de probabilidad con distribución F . Para $a < b$, se define la densidad truncada:

$$f^*(x) = \begin{cases} \frac{f(x)}{F(b) - F(a)} & \text{si } a \leq x \leq b \\ 0 & \text{en cualquier otro caso} \end{cases}$$

y, la correspondiente distribución de probabilidad acumulada truncada es:

$$F^*(x) = \begin{cases} 0 & \text{si } x < a \\ \frac{F(x) - F(a)}{F(b) - F(a)} & \text{si } a \leq x \leq b \\ 1 & \text{si } b < x \end{cases}$$

Con lo cual, un algoritmo para generar X con distribución truncada F^* es:

- (1)- Generar una variable aleatoria $U(0,1)$, U .
- (2)- Definir $V = F(a) + [F(b) - F(a)]U$. Obsérvese que esta distribuida uniformemente entre $F(b)$ y $F(a)$
- (3)- Sea $X = F^{-1}(V)$

6.3.3- Composición

Cuando la función de distribución acumulada F , de la que se quiere muestrear, puede expresarse como una combinación de otras funciones distribución F_1, F_2, \dots de la forma:

$$F(x) = \sum_{j:1}^{\infty} p_j F_j(x)$$

donde $p_j \geq 0$ (posiblemente, a partir de cierto valor de j , los coeficientes p_j serán cero), $\sum_{j:1}^{\infty} p_j = 1$ y cada F_j es una distribución de probabilidad acumulada, puede aplicarse el *algoritmo de composición*:

- (1)- Generar un entero positivo aleatorio j tal que: $P\{J = j\} = p_j$ para $j:1,2,\dots$. Puede considerarse como escoger la distribución la distribución F_j con probabilidad p_j .
- (2)- Dado que $J=j$, generar X de la distribución F_j .

6.3.4- Convolución

En algunas distribuciones importantes, la variable aleatoria deseada puede expresarse como suma de otras variables aleatorias IID que pueden generarse más fácilmente que generar directamente X . Supongamos que Y_1, Y_2, \dots, Y_m son variables aleatorias IID con distribución G y que $Y_1 + Y_2 + \dots + Y_m$ tiene la misma distribución que X , F ; entonces escribimos

$$X = Y_1 + Y_2 + \dots + Y_m$$

el algoritmo, llamado de *convolución*, para obtener X es:

- (1)- Generar las variables aleatorias IID Y_1, Y_2, \dots, Y_m , cada una con distribución G .
- (2)- Asignar $X = Y_1 + Y_2 + \dots + Y_m$.

A continuación enunciaremos algunos algoritmos para generar variables aleatorias X a partir de varias de las *distribuciones continuas* más comunes.

6.3.5- Generación de variables aleatorias continuas

Uniforme, $U(a,b)$

Algoritmo de la transformación inversa:

- (1)- Generar U , distribuida $U(0,1)$
- (2)- Asignar $X = a + (b - a)U$

Exponencial, $\exp o(\beta)$

Algoritmo de la transformación inversa:

- (1)- Generar U , distribuida $U(0,1)$
- (2)- Asignar $X = -\beta \ln(U)$

Normal, $N(\mu, \sigma^2)$

Algoritmo de Box y Muller para generar dos variables $N(0,1)$:

- (1)- Generar dos variables aleatorias IID, U_1 y U_2 , distribuidas $U(0,1)$.
- (2)- Asignar $X_1 = \sqrt{-2 \ln U_1} \cos(2\pi U_2)$ y $X_2 = \sqrt{-2 \ln U_1} \sin(2\pi U_2)$. X_1 y X_2 son dos variables $N(0,1)$ IID.

Dada X distribuida $N(0,1)$, obtenemos X' , distribuida $N(\mu, \sigma^2)$, mediante: $X' = \mu + \sigma X$.

Triangular, $\text{triang}(a,b,c)$

Algoritmo de la transformación inversa para generar X , distribuida $\text{triang}(0,1,c)$:

- (1)- Generar U , distribuida $U(0,1)$
- (2)- Asignar $X = \begin{cases} \sqrt{cU} & \text{si } U \leq c \\ 1 - \sqrt{(1-c)(1-U)} & \text{en cualquier otro caso} \end{cases}$

Si X esta distribuida $\text{triang}\left(0,1,\frac{c-a}{b-a}\right)$, entonces, $X' = a + (b-a)X$ esta distribuida $\text{triang}(a,b,c)$.

Distribución empírica.

Si se dispone de las observaciones empíricas individuales. Algoritmo de la transformación inversa:

- (1)- Generar U , distribuida $U(0,1)$. Sean $P = (n-1)U$ y $I = \lfloor P \rfloor + 1$. Recuérdese que $\lfloor z \rfloor$ es el mayor entero que es menor o igual que el número real z .
- (2)- Asignar $X = X_{(I)} + (P - I + 1)(X_{(I+1)} - X_{(I)})$

Si se dispone de las observaciones empíricas agrupadas; es decir, tenemos k intervalos adyacentes $[a_0, a_1), [a_1, a_2), \dots, [a_{k-1}, a_k]$, y el intervalo j -ésimo contiene n_j observaciones.

Algoritmo de la transformación inversa:

- (1)- Generar U , distribuida $U(0,1)$.
- (2)- Encontrar el entero no negativo J ($0 \leq J \leq k-1$) tal que $G(a_J) \leq U < G(a_{J+1})$. Obsérvese que X no puede generarse en un intervalo para el cual $n_j = 0$, ya que $G(a_j) < G(a_{j+1})$. Asignar

$$X = a_J + \frac{[U - G(a_J)](a_{J+1} - a_J)}{G(a_{J+1}) - G(a_J)}.$$

6.3.6- Generación de variables aleatorias discretas

A continuación enunciaremos algunos algoritmos para generar variables aleatorias X a partir de varias de las *distribuciones discretas* más comunes.

Bernoulli, Bernoulli(b)

Algoritmo de la transformación inversa (intercambiando los papeles de U y $(1-U)$):

- (1)- Generar U , distribuida $U(0,1)$.
- (2)- Si $U \leq b$ asignar $X=1$, en caso contrario, $X=0$

Uniforme discreta, $DU(i,j)$

Algoritmo de la transformación inversa:

- (1)- Generar U , distribuida $U(0,1)$.
- (2)- Asignar $X = i + \lfloor (j - i + 1)U \rfloor$

Binomial, $\text{bin}(t,b)$

La suma de t variables aleatorias IID distribuidas Bernoulli(b), tiene una distribución $\text{bin}(t,b)$.

Aplicando el algoritmo de convolución:

- (1)- Generar las variables aleatorias Bernoulli(b) IID, Y_1, Y_2, \dots, Y_t .
- (2)- Asignar $X = Y_1 + Y_2 + \dots + Y_t$

Geométrica, $\text{geom}(b)$

Algoritmo de la transformación inversa (intercambiando los papeles de U y $(1-U)$):

- (1)- Generar U , distribuida $U(0,1)$.

$$(2)- \text{Asignar } X = \left\lceil \frac{\ln U}{\ln(1-b)} \right\rceil$$

Binomial negativa, $\text{negbin}(t,b)$

La suma de t variables aleatorias IID distribuidas $\text{geom}(b)$, tiene una distribución $\text{negbin}(t,b)$.

Aplicando el algoritmo de convolución:

- (1)- Generar las variables aleatorias $\text{geom}(b)$ IID, Y_1, Y_2, \dots, Y_t .
- (2)- Asignar $X = Y_1 + Y_2 + \dots + Y_t$

Poisson, $\text{Poisson}(\lambda)$

El algoritmo se basa en la relación entre $\text{Poisson}(\lambda)$ y $\exp\left(\frac{1}{\lambda}\right)$:

- (1)- Sean $a = e^{-\lambda}$, $b=1$, $i=0$
- (2)- Generar U_{i+1} , distribuida $U(0,1)$. Reemplazar b por bU_{i+1} . Si $b < a$, asignar $X=i$; en caso contrario reemplazar i por $i+1$ y repetir el paso (2).

6.3.7- Generación de procesos de llegada

Veamos cómo generar los instantes de llegada, t_1, t_2, \dots , para un *proceso de Poisson*.

Un *proceso de Poisson estacionario* de frecuencia $\lambda > 0$ tiene la propiedad de que los intervalos de tiempo entre llegadas consecutivas, $A_i = t_i - t_{i-1}$, son variables $\exp\left(\frac{1}{\lambda}\right)$ IID. Así pues, se pueden generar los t_i 's recursivamente:

- (1)- Generar U , distribuida $U(0,1)$, independiente cualquier variable aleatoria anterior.
- (2)- Asignar $t_i = t_{i-1} - \frac{1}{\lambda} \ln U$. La recursión comienza con t_1 , sabido que $t_0 = 0$.

Para generar un *proceso de Poisson no estacionario* puede emplearse un método que presentamos para el caso en que $\lambda^* = \max_s \{\lambda(s)\}$ sea finito. Consiste en generar un proceso estacionario de Poisson con frecuencia constante λ^* , que notaremos $\{t_i^*\}$, y reducir los t_i 's eliminando cada llegada t_i^* con una probabilidad $1 - \frac{\lambda(t_i^*)}{\lambda^*}$. Así pues, es más probable aceptar t_i^* si $\lambda(t_i^*)$ es alta, dando lugar a la propiedad deseada de que las llegadas ocurran más frecuentemente en los intervalos en los cuales $\lambda(t_i^*)$ es alta. Un algoritmo equivalente, expresado de forma más conveniente, es:

- (1)- Asignar $t = t_{i-1}$
- (2)- Generar dos variables aleatorias $U(0,1)$ IID, U_1, U_2 , independientes de las generadas anteriormente.
- (3)- Reemplazar t por $t - \frac{1}{\lambda^*} \ln U_1$
- (4)- Si $U_2 \leq \frac{\lambda(t)}{\lambda^*}$, asignar $t_i = t$. En caso contrario, volver al paso (2).

En el caso en que los clientes lleguen por grupos, y que el número de clientes de cada grupo sea una variable aleatoria discreta, B_i , IID e independiente de los t_i 's, puede emplearse el siguiente algoritmo para generar el proceso de llegada:

- (1)- Generar el siguiente instante de llegada, t_i .
- (2)- Generar la variable aleatoria discreta B_i . Así pues, B_i llegan en el instante t_i .