

GENERACION DE NUMEROS ALEATORIOS

Un paso clave en simulación es tener rutinas que generen variables aleatorias con distribuciones específicas: exponencial, normal, etc. Esto es hecho en dos fases. La primera consiste en generar una secuencia de números aleatorios distribuidos uniformemente entre 0 y 1. Luego esta secuencia es transformada para obtener los valores aleatorios de las distribuciones deseadas. La primera fase es la que nos concierne ahora.

I. PROPIEDADES DESEADAS DE BUENOS GENERADORES

Veamos como operan los generadores para poder entender porque uno puede ser considerado mejor que otro. El método más común es generar el siguiente número a partir de los últimos números generados:

$$x_n = f(x_{n-1}, x_{n-2}, \dots)$$

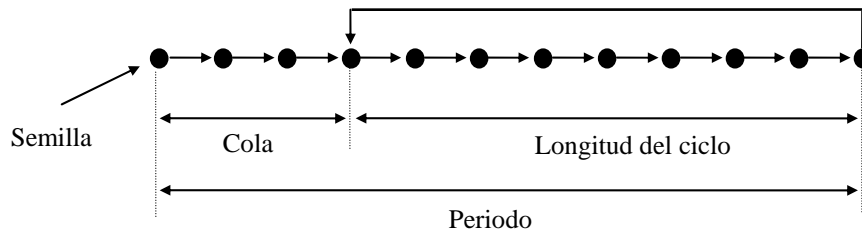
Una de estas funciones es:

$$x_n = (5x_{n-1} + 1) \bmod 16$$

Si comenzamos con $x_0 = 5$ los primeros 32 números generados son: 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5, 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5. Las x son enteros entre 0 y 15, y si las dividimos entre 16 obtenemos una secuencia de números aleatorios entre 0 y 1: 0.6250, 0.1875, 0.0000, ...

Si conocemos f podemos generar la secuencia en cualquier momento si tenemos el valor de x_0 . El valor usado para comenzar la secuencia es llamado **semilla**. Nótese que f es determinística. Dada la semilla se puede predecir con probabilidad 1 los números de la secuencia. Sin embargo, los números son aleatorios en el sentido de que pasan pruebas estadísticas de aleatoriedad y por esto son llamados **pseudo-aleatorios**. En muchos casos se prefieren estos números en vez de los completamente aleatorios ya que es necesario repetir las secuencias en distintos experimentos. Si deseamos otra secuencia simplemente cambiamos la semilla.

Nótese que en el ejemplo la secuencia tiene un ciclo y la **longitud del ciclo** es 16. Algunos generadores no repiten la parte inicial de la secuencia. Esta parte es llamada **cola**. En estos casos el **periodo** del generador es la longitud de la cola más la longitud del ciclo.



La propiedades deseadas del generador son las siguientes:

1. *Deben ser eficientes computacionalmente*: dado que típicamente se requieren varios miles de números aleatorios por corrida, el tiempo de procesador requerido para generarlos debe ser pequeño.
2. *El periodo debe ser largo*: periodos cortos limitan la longitud aprovechable de una corrida de simulación porque el reciclaje resulta en una repetición de secuencias de eventos.
3. *Los valores sucesivos deben ser independientes y uniformemente distribuidos*: la correlación entre números sucesivos debe ser pequeña y si es significativa indica dependencia.

Las primeras dos propiedades son relativamente fáciles de implementar. La tercera requiere un conjunto de pruebas estadísticas. Entre los generadores que discutiremos están:

- Generadores congruenciales-lineales
- Generadores de Fibonacci extendidos
- Generadores Combinados

II. GENERADORES CONGRUENCIALES-LINEALES (GCL)

En 1951, D. H. Lehmer descubrió que residuos de potencias sucesivas de un número tienen buenas propiedades aleatorias:

$$x_n = a^n \text{ mod } m$$

Una expresión equivalente para calcular x_n después de calcular x_{n-1} es:

$$x_n = ax_{n-1} \text{ mod } m$$

Los parámetros a y m son llamados **multiplicador** y **modulo** respectivamente. Muchos de los generadores actuales son generalizaciones de la propuesta de Lehmer y tienen la siguiente forma:

$$x_n = (ax_{n-1} + b) \text{ mod } m$$

en donde los x son enteros entre 0 y $m-1$, y las constantes a y b son no-negativas. La selección de a , b , y m afectan el periodo y la autocorrelación en la secuencia. Entre los resultados de los estudios realizados con estos generadores tenemos:

1. El modulo m debe ser grande. Dado que los x están entre 0 y $m-1$, el periodo nunca puede ser mayor que m .
2. Para que el computo de mod m sea eficiente, m debe ser una potencia de 2, es decir, 2^k . En este caso mod m puede ser obtenido truncando el resultado y tomando en k bits a la derecha.

Ejemplo

$$45 \bmod 16 = 45 \bmod 2^4 = 13$$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	1	0	1	1	0	1

13

3. Si b es diferente de cero, el periodo máximo posible m se obtiene si y solo si:

- a) Los enteros m y b son primos relativos -- no tengan factores comunes excepto el 1.
- b) Todo número primo que sea un factor de m lo es también de $a-1$.
- c) $a-1$ es un múltiplo de 4 si m es un múltiplo de 4.

Todas estas condiciones se cumplen si $m = 2^k$, $a = 4c + 1$, y b es impar, donde c , b , y k son enteros positivos. Si un generador tiene el periodo máximo posible se llama generador de **periodo completo**. Todos los generadores de periodo completo no son igualmente buenos. Son preferibles los generadores con menor autocorrelación entre números sucesivos. Por ejemplo, los dos generadores siguientes son de periodo completo, pero el primero tiene una correlación de 0.25 entre x_{n-1} y x_n , mientras que el segundo tiene una correlación despreciable de menos de 2^{-18} .

$$x_n = ((2^{34} + 1)x_{n-1} + 1) \bmod 2^{35}$$

$$x_n = ((2^{18} + 1)x_{n-1} + 1) \bmod 2^{35}$$

1. GCL multiplicativos

Los GCL presentados anteriormente son GCL mixtos. Si el incremento b es cero, no hay adición y el generador es llamado **GCL multiplicativo** y tienen la forma:

$$x_n = ax_{n-1} \bmod m$$

Es obvio que estos son más eficientes que los mixtos. Eficiencia adicional puede ser obtenida tomando $m = 2^k$. Por lo tanto hay dos tipos de GCL multiplicativos dependiendo si $m = 2^k$ o no.

2. GCL multiplicativos con $m = 2^k$.

El argumento a favor de usar $m = 2^k$ esta en la eficiencia de la operación mod. Sin embargo estos generadores no son de periodo completo. El máximo periodo posible para estos generadores es un cuarto del periodo completo: 2^{k-2} , y se obtiene si el multiplicador es de la forma $8i \pm 3$ y la semilla es impar. A pesar de esto, un cuarto de periodo máximo posible puede ser suficiente para muchas aplicaciones.

Ejemplo

Consideremos el siguiente GCL multiplicativo:

$$x_n = 5x_{n-1} \pmod{2^5}$$

Si $x_0 = 1$, obtenemos la secuencia 5, 25, 29, 17, 21, 9, 13, 1, 5, ..., con periodo $8 = 32/4$. Si cambiamos $x_0 = 2$, la secuencia es 2, 10, 18, 26, 2, 10, ..., con periodo 4. Para ver que sucede si el multiplicador no es de la forma $8i \pm 3$, consideremos:

$$x_n = 7x_{n-1} \pmod{2^5}$$

Si $x_0 = 1$, obtenemos la secuencia 1, 7, 17, 23, 1, 7, ..., y vemos que ambas condiciones son necesarias para obtener el periodo máximo.

3. GCL multiplicativos con $m \neq 2^k$.

Una solución para los periodos pequeños es usar un modulo m que sea número primo. Con un multiplicador a adecuado se puede obtener un periodo de $m - 1$, que es casi el máximo periodo posible. Note que en este caso x_n nunca puede ser cero y su valor esta entre 1 y $m - 1$. Todo GLC multiplicativo con periodo $m - 1$ se dice que es de periodo completo.

Se puede demostrar que un GLC multiplicativo es de periodo completo si y solo si el multiplicador a es una raíz primitiva del modulo m ; a es una raíz primitiva de m si y solo si $a^n \pmod{m} \neq 1$ para $n = 1, 2, \dots, m-2$.

Ejemplo

Consideremos el siguiente GCL multiplicativo:

$$x_n = 3x_{n-1} \pmod{31}$$

Si $x_0 = 1$, obtenemos la secuencia 1, 3, ..., con periodo 30 y por lo tanto es de periodo completo. Si usamos $a = 5$, obtenemos la secuencia 1, 5, 25, 1, ... que es de periodo 3. Note que 3 es una raíz primitiva de 31 ya que el menor entero positivo de n tal que $3^n \pmod{31} = 1$ es $n = 30$, y 5 no lo es ya que $5^3 \pmod{31} = 1$.

El siguiente es un GCL multiplicativo es de periodo completo:

$$x_n = 7^5 x_{n-1} \pmod{(2^{31} - 1)}$$

Este generador data de principios de los 70 y fue usado en diversos sistemas de IBM, el sistema operativo PRIMOS, la librería científica IMSL, y en los lenguajes de simulación SIMAN y ARENA entre otros. Su ciclo es de longitud $2^{31}-2 \sim 2.1 \times 10^9$. Este generador es bastante aceptado ya que ha sido bien probado y produce buenas secuencias de números.

Las propiedades del GCL son garantizadas solo si los cálculos son realizados exactamente y sin errores

de redondeo. Hay que usar aritmética entera sin desbordamientos. Hay que tener cuidado si se usan lenguajes como BASIC ya que los cálculos son hechos usando números reales donde el truncamiento puede reducir considerablemente el periodo.

El segundo problema está en que ax_{n-1} puede exceder el máximo entero permitido en el sistema, causando desbordamiento de enteros. Una solución está basada en la siguiente identidad:

$$ax \bmod m = g(x) + mh(x)$$

donde

$$g(x) = a(x \bmod q) - r(x \operatorname{div} q)$$

y

$$h(x) = (x \operatorname{div} q) - (ax \operatorname{div} m)$$

donde

$$q = m \operatorname{div} a \quad \text{y} \quad r = m \bmod a.$$

Se puede demostrar que para todos los x entre 1 y $m-1$, hay menos de $m-1$ expresiones involucradas en calcular $g(x)$. Además, si $r < q$, $h(x)$ es 0 o 1 y puede ser inferido de $g(x)$: $h(x)$ es 1 si y solo si $g(x) < 0$.

Ejemplo

Consideremos el siguiente GCL multiplicativo:

$$x_n = 7^5 x_{n-1} \bmod (2^{31} - 1)$$

o

$$x_n = 16807 x_{n-1} \bmod 2147483647$$

El producto ax_{n-1} puede llegar a ser $16807 \times 2147483646 \approx 1.03 \times 2^{45}$. Una implementación directa producirá un desborde de enteros excepto que el sistema use enteros de 46 bits o más.

En nuestro caso:

$$a = 16807$$

$$m = 2147483647$$

$$q = m \operatorname{div} a = 2147483647 \operatorname{div} 16807 = 127773$$

$$r = m \bmod a = 2147483647 \bmod 16807 = 2836.$$

El programa que se muestra a continuación implementa correctamente el generador en sistemas en los cuales el máximo entero sea $2^{31}-1$ o mayor. Para probar si la implementación es correcta calcule $x_{10.000}$ comenzando con $x_0 = 1$ y debe dar 1043618065.

Programa	Salida
<pre> program prueba_random; var i,x: integer; ran: real; function random(var x:integer): real; const a = 16807; { multiplicador } m = 2147483647; { modulo } q = 127773; { m div a } r = 2836; { m mod a} begin x := a*(x mod q) - r*(x div q); if x < 0 then x := x + m; random := x/m; end; begin x:=1; { semilla } for i:=1 to 10000 do ran:=random(x); writeln(x); end. </pre>	1043618065

III. GENERADORES DE FIBONACCI EXTENDIDOS

Una secuencia de Fibonacci $\{x_n\}$ se genera por la siguiente relación:

$$x_n = x_{n-1} + x_{n-2}$$

Se puede intentar usar un generador de la forma:

$$x_n = (x_{n-1} + x_{n-2}) \bmod m$$

sin embargo esta secuencia no tiene buenas propiedades aleatorias y en particular tiene alta correlación serial. El siguiente generador, que sigue este enfoque, pasa la mayoría de las pruebas estadísticas:

$$x_n = x_{n-5} + x_{n-17} \bmod 2^k$$

Para implementar este generador se pueden usar 17 localidades de memoria $L[1], \dots, L[17]$ las cuales son inicializadas con 17 enteros que no sean todos pares. Fijamos i y j en 5 y 17 respectivamente, y el siguiente procedimiento es ejecutado para obtener los números aleatorios:

```

x := L[i] + L[j];
L[i] := x;
i:= i - 1; if i = 0 then i := 17;
j:= j - 1; if j = 0 then j := 17;

```

La adición en la primera línea es automáticamente mod 2^k en máquinas de k -bits y aritmética de complemento a 2. El periodo del generador es $2^k (2^{17} - 1)$ que es considerablemente mayor al que se puede obtener con GCL.

IV. GENERADORES COMBINADOS

Es posible combinar generadores para obtener “mejores” generadores. Algunas de las técnicas usadas son:

1. *OR-exclusivo de números aleatorios de dos o más generadores.* Esta técnica es similar a la anterior excepto que la suma es reemplazada por un or-exclusivo bit por bit. Se ha demostrado que esta técnica aplicada a números ligeramente aleatorios puede ser usada para generar números con mayor aleatoriedad.
2. *Barajeo.* Usa una secuencia como un índice para decidir que número generado por otra secuencia será retornado. Por ejemplo, uno de estos algoritmos usa un arreglo de tamaño 100 que contiene números de una secuencia aleatoria x_n . Para generar un número aleatorio se genera un número aleatorio y_n (entre 0 y $m-1$) para obtener el índice $i = 1 + 99y_n / (m-1)$. El valor del i -ésimo elemento del arreglo es devuelto. Un nuevo valor x_n es calculado y almacenado en la i -ésima localidad.

IV. GENERADORES CONGRUENCIALES LINEALES COMBINADOS

Aunque estos generadores caen dentro de la sección anterior, su relevancia amerita su discusión por separado.

Consideremos nuevamente el generador $x_n = 7^5 x_{n-1} \pmod{(2^{31} - 1)}$. En la medida que las computadoras se han vuelto más rápidas, la longitud de su ciclo se ha tornado inadecuada ya que se corre el riesgo de que, en unas cuantas horas de simulación, la secuencia se agote y se repita varias veces trayendo como consecuencia serias dudas en cuanto a la validez de los resultados.

Supongamos que tenemos una máquina con un procesador de 1GHz. Esto son 10^9 ciclos o tics del reloj por segundo. Supongamos también que el procesador es capaz de generar un número aleatorio por ciclo (esto es muy optimista ya que en realidad se requieren varios ciclos para producir un número aleatorio debido a las operaciones involucradas). Bajo estas condiciones se agotaría la secuencia en $2.1 \times 10^9 / 10^9 = 2.1 \text{seg}$. Por supuesto que será en más tiempo, pero se observa que efectivamente esta secuencia es fácilmente agotable durante una simulación.

Una manera para conseguir generadores con periodos más largos consiste en sumar números aleatorios de dos o más generadores. El siguiente resultado debido a L'Ecuyer (<http://www.iro.umontreal.ca/~lecuyer/>) muestra como esto puede ser realizado.

Si $W_{i,1}, W_{i,2}, \dots, W_{i,k}$ son variables aleatorias discretas independientes, no necesariamente idénticamente distribuidas, y una de ellas, digamos la primera $W_{i,1}$, es entera y uniforme en 0 a m_1-1 , entonces

$$W_i = \left(\sum_{j=1}^k W_{i,j} \right) \pmod{(m_1 - 1)}$$

es uniforme sobre los enteros 0 a m_j-2 .

Veamos como esto puede ser usado para combinar generadores. Sean $X_{i,1}, X_{i,2}, \dots, X_{i,k}$ la i -ésima salida de k generadores congruenciales multiplicativos diferentes y donde el j -ésimo generador tiene modulo primo m_j y su multiplicador a ha sido seleccionado de forma tal que tenga periodo máximo $m_j - 1$. L'Ecuyer sugiere generadores combinados de la forma:

$$X_i = \left(\sum_{j=1}^k (-1)^{j-1} X_{i,j} \right) \text{ mod } (m_1 - 1)$$

$$R_i = \begin{cases} \frac{X_i}{m_1} & X_i > 0 \\ \frac{m_1 - 1}{m_1} & X_i = 0 \end{cases}$$

y el periodo máximo posible es $P = \frac{(m_1 - 1)(m_2 - 1)\dots(m_k - 1)}{2^{k-1}}$.

Uno de estos generadores combinados usa:

$$x_n = 40014x_{n-1} \text{ mod } (2^{31}-85)$$

$$y_n = 40692y_{n-1} \text{ mod } (2^{31}-249)$$

para obtener

$$w_n = (x_n - y_n) \text{ mod } (2^{31}-86)$$

$$R_n = \begin{cases} \frac{w_n}{(2^{31} - 85)} & w_n > 0 \\ \frac{(2^{31} - 86)}{(2^{31} - 85)} & w_n = 0 \end{cases}$$

con un periodo aproximado $P = \frac{(2^{31} - 85)(2^{31} - 249)}{2^{2-1}} = 2.3 \times 10^{18}$. En este caso se necesitan dos semillas x_o y y_o para comenzar la generación. Haciendo cálculos y supuestos similares a los anteriores, tendríamos que la secuencia se consumiría en $\frac{2.3 \times 10^{18}}{10^9 \times 60 \times 60 \times 24 \times 365} = 72.93$ años.

Obsérvese que los números que producen los dos generadores básicos (x_n y y_n) están entre 0 y 2^{31} aproximadamente, y la secuencia w_n solo se repite cuando estos generadores básicos, ambos, coinciden nuevamente en los valores usados para arrancarlos (las semillas: x_o y y_o), es decir, la secuencia se repite solo cuando los generadores básicos se sincronizan en $x_n = x_o$ y $y_n = y_o$. No es como en el caso de un solo generador en que al repetirse un número de la secuencia esta también se repita. Ahora los números pueden repetirse sin que implique que la secuencia se repita.

Con un periodo aproximado de 3.1×10^{57} . Haciendo nuevamente los supuestos y cálculos anteriores tenemos $\frac{3.1 \times 10^{57}}{10^9 \times 60 \times 60 \times 24 \times 365} = 9.83 \times 10^{40}$ años.

Este generador necesita 6 semillas ($Z_{1,0}, Z_{1,1}, Z_{1,2}, Z_{2,0}, Z_{2,1}, Z_{2,2}$), y a pesar de que es un poco más costoso computacionalmente que un generador simple, con las velocidades de la máquinas de hoy en día no hay ningún problema en usarlos. L'Ecuyer determino sus parámetros después de un extenso estudio.

V. ALGUNOS GENERADORES DE NUMEROS ALEATORIOS

- Un generador GCL muy popular y que ya hemos mencionado varias veces es:

$$x_n = 7^5 x_{n-1} \pmod{(2^{31} - 1)}$$

Este es usado en el sistema SIMPL/I de IBM (1972), APL de IBM (1971), el sistema operativo PRIMOS de Prime Computer (1984), y la librería científica de IMSL (1987). Tiene buenas propiedades aleatorias y es recomendado como un estándar mínimo.

- Un estudio de generadores multiplicativos GCL con modulo $m = 2^{31} - 1$ que comparo su eficiencia y aleatoriedad, recomienda los dos siguientes como los mejores:

$$x_n = 48.271 x_{n-1} \pmod{(2^{31} - 1)}$$

$$x_n = 69.621 x_{n-1} \pmod{(2^{31} - 1)}$$

- El siguiente generador es usado en SIMSCRIPT II.5 y en FORTRAN DEC-20:

$$x_n = 630.360.016 x_{n-1} \pmod{(2^{31} - 1)}$$

- El siguiente generador es usado en el sistema Pascal de la Universidad de Sheffield para computadores Prime:

$$x_n = 16.807 x_{n-1} \pmod{2^{31}}$$

Dado que 16.807 no es de la forma $8i \pm 3$, este generador no tiene el periodo máximo posible de 2^{31-2} . También es usado en la subrutina UNIFORM del paquete estadístico SAS , pero una técnica de barajeo es usada para mejorar la aleatoriedad.

- SIMULA en UNIVAC usa:

$$x_n = 5^{13} x_{n-1} \pmod{2^{35}}$$

Algunos autores dicen que no tiene buenas propiedades aleatorias.

- El sistema operativo UNIX soporta en siguiente GCL mixto:

$$x_n = (1.103515.245 x_{n-1} + 12.345) \pmod{2^{32}}$$

Diseñar nuevos generadores parece muy simple, pero muchos generadores propuestos por expertos estadísticos fueron encontrados deficientes. Por lo tanto *es mejor usar un generador que ha sido extensamente probado en vez de inventar uno nuevo.*

VI. SELECCIÓN DE LA SEMILLA

En principio la semilla no debería afectar los resultados de la simulación. Sin embargo, una mala combinación de semilla y generador pueden producir conclusiones erróneas.

Si el generador es de periodo completo y solo se requiere una variable aleatoria, cualquier semilla es buena. Hay que tener especial cuidado en simulaciones que requieren números aleatorios para más de una variable (**simulaciones de secuencias múltiples**), que es la mayoría de los casos. Por ejemplo, la simulación de una cola simple requiere generar llegadas y servicios aleatorios y requiere dos secuencias de números aleatorios.

Recomendaciones para la selección de las semillas:

1. *No use cero.* Cero funciona para generadores GCL mixtos pero hace que los multiplicativos se queden en cero.
2. *Evite valores pares.* Si un generador no es de periodo completo (por ejemplo GCL multiplicativo con modulo $m = 2^k$) la semilla debe ser impar. En otros casos no importa.
3. *No subdivida una secuencia.* Usar una única secuencia para todas las variables es un error común. Por ejemplo, en la secuencia $\{u_1, u_2, \dots\}$ generada a partir de la semilla u_0 , el analista usa u_1 para generar el tiempo entre llegadas, u_2 para el tiempo de servicio, etc. Esto puede resultar en una fuerte correlación entre las variables.
4. *Use secuencias que no se solapen.* Cada secuencia requiere su semilla. Si la semilla es tal que hace que dos secuencias se solapen, habrá correlación entre las secuencias y estas no serán independientes. Consideremos el ejemplo trivial de iniciar las dos secuencias de una cola simple con la misma semilla, lo cual haría las secuencias idénticas e introduciría una fuerte correlación positiva entre los tiempos entre llegadas y los tiempos de servicio. Esto puede llevar a conclusiones erróneas.

Hay que seleccionar las semillas de forma tal que las secuencias no se solapen. Si $\{u_1, u_2, \dots\}$ generada a partir de la semilla u_0 , y necesitamos por ejemplo 10.000 tiempos entre llegadas, 10.000 tiempos de servicios, etc., podemos seleccionar u_0 como la semilla de la primera secuencia, $u_{10.000}$ para la segunda, $u_{20.000}$ para la tercera, etc. Los u_n se pueden determinar mediante un programa de prueba que llama al generador o se pueden calcular directamente para generadores GCL mixtos o multiplicativos ($b = 0$) con la formula siempre que los cálculos sean exactos:

$$x_n = a^n x_0 + \frac{b(a^n - 1)}{a - 1} \mod m$$

5. *Reuse semillas en repeticiones sucesivas.* Si el experimento es replicado varias veces, la secuencia no necesita ser reinicializada y se puede usar la semilla dejada en la replicación previa.
6. *No use semillas aleatorias.* Semillas aleatorias, como por ejemplo la hora del día, causan dos problemas:
 - La simulación no puede ser reproducida.
 - No se puede garantizar que secuencias múltiples no se solapen.

VII. MITOS SOBRE LA GENERACIÓN DE NÚMEROS ALEATORIOS

Estos son algunos mitos que analistas desinformados pueden creer ciertos:

1. *Un conjunto de operaciones complejas lleva a resultados aleatorios.* Resultados difíciles de predecir no necesariamente son aleatorios. Es mejor usar operaciones simples cuya aleatoriedad pueda ser evaluadas analíticamente.
2. *Una sola prueba como la chi-cuadrado es suficiente para probar si el generador es bueno.* Obviamente la secuencia 0, 1, 2, ..., $m-1$ no es aleatoria, sin embargo pasa la prueba chi-cuadrado. En general se debe evitar inventar generadores a no ser que se sea un experto estadístico.
3. *Números aleatorios son impredecibles.* Un secuencia realmente aleatoria debe ser completamente impredecible. Este no es el caso para los generadores de números pseudo-aleatorios. Dada una escasa secuencia de números de un GCL, fácilmente se pueden determinar los parámetros a , b , y m , y predecir toda la secuencia. Por lo tanto, GCL no son aptos en aplicaciones criptográficas en las cuales se requieren secuencias impredecibles.
4. *Algunas semillas son mejores que otras.* Esto puede ser cierto en algunos casos, por ejemplo en

$$x_n = (9806x_{n-1} + 1) \bmod (2^{17} - 1)$$

si se usa como semilla 37911 el generador se queda pegado en este número. Generadores que tengan restricciones respecto a la semilla deben ser evitados.

5. *Implementación exacta no es relevante.* El periodo y la aleatoriedad de un generador se garantizan solo si es implementado exactamente sin desbordes, truncamientos o redondeos.