Generación de números aleatorios

Jhon Jairo Padilla Aguilar, PhD.

Necesidad

- Un elemento clave para el desarrollo de un simulador es la generación de valores aleatorios para variables con una determinada distribución de probabilidad (variables aleatorias).
- La introducción de variables aleatorias en los modelos de simulación permiten recrear situaciones que no están completamente controladas, o que dependen de algún factor aleatorio.
- Por ejemplo, como vimos en temas anteriores, el tiempo entre llegadas de clientes en una estación de servicio suele modelarse como una distribución exponencial.

Principio básico

- Un generador de valores para una variable aleatoria se consigue en dos pasos:
  - Primero tenemos que definir un generador de números aleatorios distribuidos uniformemente entre 0 y 1.
  - Después, transformaremos esta secuencia para generar los valores de la variable aleatoria deseada.

Requisitos de los generadores

- Un requerimiento importante que deben cumplir los generadores de números aleatorios es **que las series generadas sean reproducibles**, de tal modo que en distintos experimentos podamos utilizar exactamente la misma secuencia de números aleatorios para reproducir las mismas condiciones.
- El generador **debe permitir la generación de varias secuencias separadas de números aleatorios**. Una *secuencia* es un segmento de los números producidos por el generador: una secuencia termina donde comienza la siguiente.
- De este modo, puede dedicarse una secuencia diferente para cada fuente de aleatoriedad del modelo. Por ejemplo, una secuencia para los intervalos entre llegadas de un tipo de entidad, otra secuencia para los tiempos de proceso de un determinado recurso, etc.

Uso de los generadores

- En la actualidad, prácticamente todos los lenguajes de programación proveen alguna función para generar números aleatorios.
- Aunque podríamos pensar que estos son suficientes para realizar nuestras simulaciones, es necesario que nos aseguremos de la buena calidad de estos generadores.
- El empleo de un generador que produzca series con fuertes dependencias o correlacciones puede conducir a simulaciones incorrectas y a conclusiones totalmente falsas.

Principio básico

- Un generador de números aleatorios consiste en una función que devuelve los valores de una secuencia de números reales, (u_1, u_2, . . . , u_n), donde cada u_i está en el rango [0, 1].
- El primer elemento de la secuencia se le denomina **semilla inicial** de la serie, y a partir de ella debe ser posible generar el resto de la secuencia para que ésta sea reproducible.

Propiedades de un generador

- Entre las propiedades que debe cumplir un buen generador para realizar simulaciones, destacaríamos las siguientes:
  - Los valores u_i deben ser independientes y estar idénticamente distribuidos (IID).
  - La secuencia generada debe ser bastante larga, con el fin de permitir simulaciones largas y/o con muchas variables aleatorias.
  - Debe computarse muy eficientemente, ya que cada simulación podría requerir la generación de una cantidad considerable de números aleatorios, del orden de millones.

Propiedades de un generador

- La **condición de uniformidad** debe cumplirse además para todas las subsecuencias de tamaño k de la secuencia generada. Es decir:
  - los u_i deben distribuirse uniformemente en [0, 1] los pares (u_i, u_{i+1}) deben distribuirse uniformemente en el plano [0, 1] × [0, 1] los tríos (u_i, u_{i+1}, u_{i+2}) deben distribuirse uniformemente en el cubo $[0, 1]^3$, etc
- Asegurar la k-uniformidad de las secuencias es crucial para los experimentos de simulación. Con ella se evitan correlaciones entre distintas variables que compartan una misma secuencia.

Propiedades de un generador

- **Independencia**:
- Una serie es independiente si no puede ser especificada mediante un algoritmo que requiera menos bits que la serie en sí.
  - Por ejemplo, la serie de bits: 01010000001010011011 . . . es independiente si la forma más rápida de describirla es simplemente escribirla.
  - Por el contrario, la serie de bits: 0101010101010101 . . . puede ser descrita mediante un algoritmo más corto que la serie en sí.

Tipos de Generadores

Físicos

- Son dispositivos físicos que emplean fuentes externas: la desintegración de un material radioactivo o el ruido eléctrico, para generar números aleatorios.

Aritméticos

- Son algoritmos deterministas que son ejecutados mediante el computador.

Generadores físicos

- Los más frecuentemente usados están basados en circuitos eléctricos dotados de una fuente de ruido (frecuentemente una resistencia o un diodo semiconductor) que es amplificada, muestreada y comparada con una señal de referencia para producir secuencias de bits. Por ejemplo, si el ruido es menor que la referencia se produce un 0 y si es mayor un 1.
- Frecuentemente, en las secuencias de bits as´ı obtenidas los 1's no tienen la misma probabilidad que los 0's, con lo cual deben ser post-procesadas para obtener secuencias de bits aleatorios.
- Para su aplicación, estos bits aleatorios se unen para formar bytes, números enteros o números reales, según sea preciso.

Generadores aritméticos

- Los buenos generadores aritméticos producen secuencias de números que son indistinguibles de realizaciones independientes de variables uniformes. Existen dos grandes familias de generadores aritméticos: los generadores *lineales* y los *no lineales*.
- En simulación los generadores más comúnmente usados son los lineales.

Generadores congruenciales lineales

- Los generadores congruenciales fueron descubiertos por Lehmer en 1951 cuando observó que los residuos de las potencias sucesivas de un número sugieren un comportamiento aleatorio.
- La primera propuesta de Lehmer consistió en la siguiente secuencia:

$$x_n = a^n \ mod \ m$$

- Es decir, el número n-ésimo de la secuencia se obtiene dividiendo la potencia n-´esima de un entero a por un entero m y tomando el resto. La expresión anterior es equivalente a la siguiente:

$$x_n = a\, x_{n-1} \ mod \ m$$

- De este modo se puede obtener cada elemento de la secuencia a partir del elemento anterior, y en primer lugar de un valor inicial x_0 denominado semilla.

Generadores congruenciales lineales

- Muchos de los generadores propuestos actualmente son una generalización del propuesto por Lehmer, donde se incluye un sesgo b en la expresión anterior:

$$x_n = (a\, x_{n-1} + b) \bmod m$$

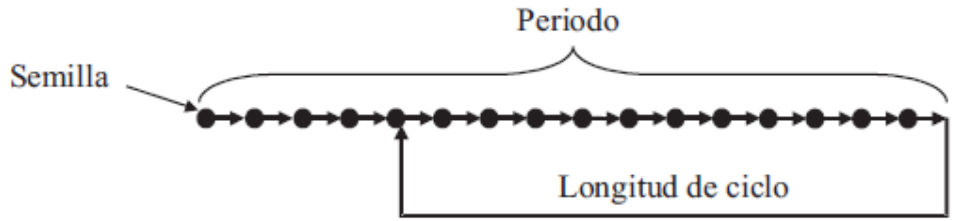
- La secuencia así obtenida consiste en una serie de números enteros comprendidos entre 0 y m − 1.
- Los parámetros del generador a, b y m deben de ser números enteros positivos.
- Por otro lado, cuando b = 0 diremos que el generador es multiplicativo.

Selección de parámetros

- Para obtener una secuencia de n´umeros entre 0 y 1 a partir de un GCL, bastará con dividir cada x_n por el módulo m de la serie:
- $u_n = x_n/m$
- A pesar de su simplicidad, la selección adecuada de las constantes (a, b,m) permitirá obtener secuencias largas y aleatorias de un modo muy eficiente.

Selección de parámetros

- Como se muestra en la figura, los GLCs producen secuencias cíclicas.
- Una de las propiedades que nos interesa en Simulación es que el periodo de repetición de las secuencias sea lo más grande posible.
- Dicho periodo vendrá determinado por los parámetros del generador.



Condiciones para máximo período

- Se ha demostrado que el máximo periodo se alcanza cuando se dan las siguientes condiciones:
- b diferente de 0
- mcd(b,m) = 1 (primos relativos)
- a = *1 mod p ;* para cada factor primo *p* de *m*
- a = *1 mod 4 ;* si 4 divide a *m*

Condiciones para máxima Eficiencia

- Además de la longitud de los ciclos, en simulación también nos interesa que el generador sea muy eficiente. Dado que los GLCs multiplicativos (b = 0) son los más eficientes de computar, a partir de ahora sólo estudiaremos este tipo de generadores.

Eficiencia

- Es fácil ver que la operación más costosa en un GLC es la operación del módulo (o resto) (m suele ser un número muy grande). Si tomamos m = 2^{β} la operación de resto resulta obvia, ya que basta retener los últimos bits del producto $a \cdot x_i$.
- Sin embargo, la longitud máxima de este tipo de generadores es m/4, y se da cuando la semilla es impar y la constante a tiene la forma $8 \cdot i \pm 3$.

Longitud del período

- La longitud del periodo de un GLC multiplicativo puede llegar a ser m−1 cuando m es un número primo (en este caso los valores de la serie estarán entre 1 y m−1, nunca valdrán cero). Además, el valor de a tiene que ser cuidadosamente seleccionado.
- A este respecto, se ha demostrado que si a es una raíz primitiva de m, entonces se alcanza el periodo máximo m − 1. Esto ocurre cuando $a_n \bmod m$ diferente de 1 para n = 1, 2, . . . ,m − 2.

Ejemplo

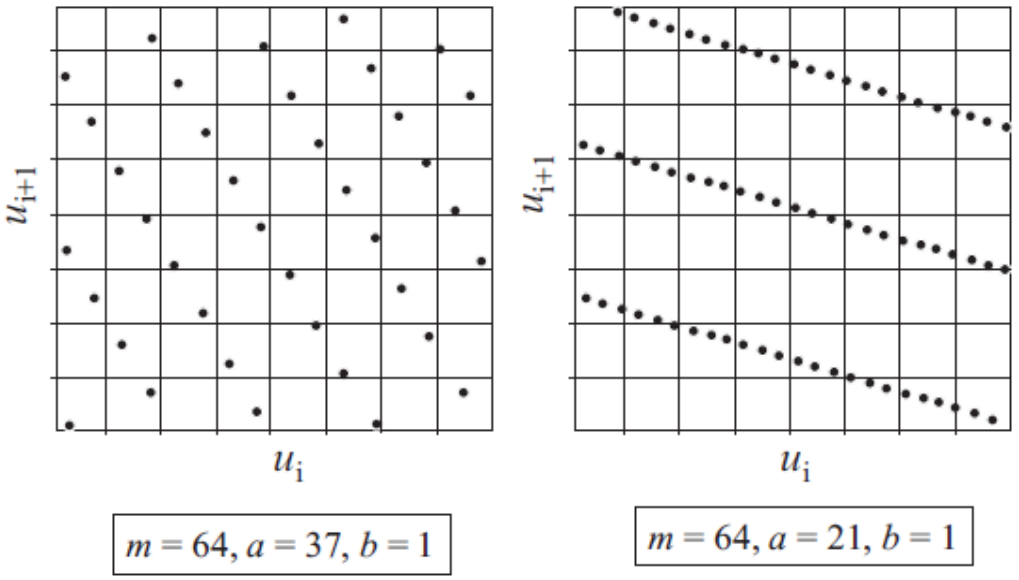
- Un ejemplo de GLC multiplicativo con periodo máximo m − 1 es el siguiente:
- a = 75 = 16807
- b = 0
- m = 231 − 1 = 2147483647
- Este ejemplo, de hecho, es el que se ha tomado como estándar para la generación de números aleatorios, debido principalmente a la calidad de las series obtenidas.

Calidad de la serie generada

- Un último factor que influye en la elección de los parámetros de un GLC es la calidad de la serie generada.
- La calidad de una serie se mide por la uniformidad e independencia de los valores de la serie.
- A este respecto, una propiedad inherente de los generadores GLC es que producen una estructura reticular cuando se toman subsecuencias de la serie generada.

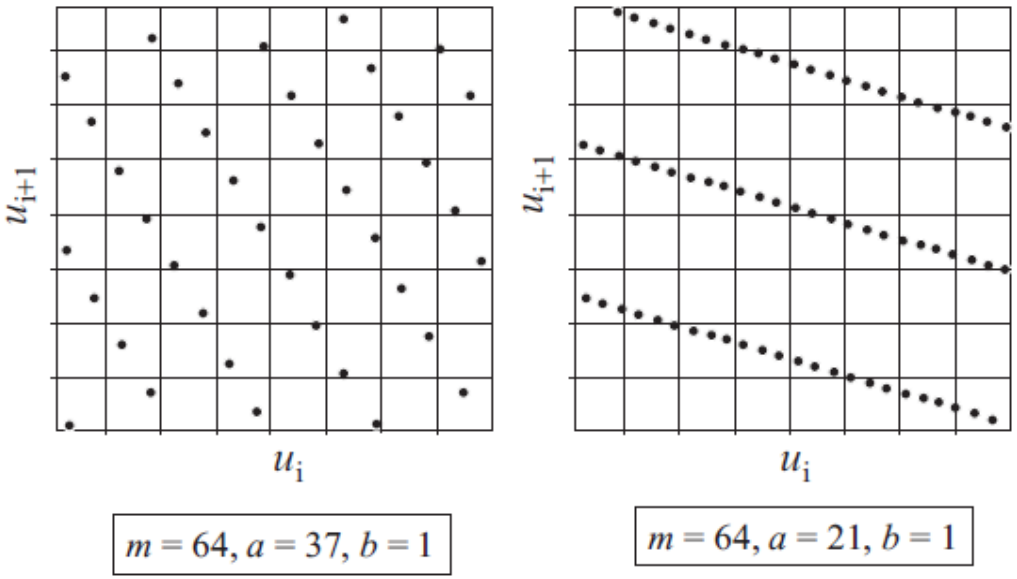
Calidad de la serie generada

- Un ejemplo típico de estructura reticular se muestra en la figura, donde se consideran los pares de números consecutivos de la serie (u_i, u_{i+1}).
- En esta estructura reticular pueden identificarse una serie de líneas (unas con pendientes positivas y otras con pendientes negativas) donde se sitúan todos los pares de la serie.



Calidad de la serie generada

- Dependiendo de la distancia entre estas líneas, los pares se distribuyen más o menos uniformemente en el plano.
- Generalmente, cuanto mayor sea la distancia, peor será el generador.
- Esta distancia viene determinada por el parámetro *a*. Así pues, el valor de *a* va a determinar la uniformidad de los valores, y su selección es crítica para la calidad de la serie final.



Implementación de los GLC

- La implementación de un GLC debe realizarse con sumo cuidado, ya que los resultados teóricos obtenidos con unos parámetros determinados podrían no ser ciertos si la implementación no realiza los cálculos exactos.
- En la implementación debe prestarse especial atención a los redondeos y a los desbordamientos en determinadas operaciones.
- En el generador GLC anterior, puede verse que el producto $a \cdot x_i$ puede llegar a $1{,}03 \cdot 2^{45}$, lo cual producirá un desbordamiento, a no ser que se use un tipo entero de al menos 46 bits.
- Para estos casos, debemos reformular el cálculo del GLC para evitar el desbordamiento.

Implementación de un GLC

- Una posible solución la propone Schrage (ver [Raj Jain, 1991]), y se resume en el siguiente fragmento de código escrito en Python:

```
int random(x):
    a = 16807
    m = 2147483647
    q = 127773          #m div a
    r = 2836            #m mod a
    x_div_q = x / q
    x_mod_q = x % q
    x_new = a*x_mod_q - r*x_div_q
    if x_new>0:
        x = x_new
    else:
        x = x_new + m
    return x
```

- Para comprobar que el generador produce el resultado esperado, el valor para x10000 debe ser 1043618065 con x0 = 1.

Generación de variables aleatorias

- Hasta el momento se ha visto cómo generar números aleatorios con distribución uniforme
- En simulación se requiere generar números aleatorios con otras distribuciones típicas.
- A continuación veremos cómo generar números aleatorios con otras distribuciones diferentes de la uniforme.

Método de Inversión

- Este método se basa en la función inversa de la función de distribución de la variable aleatoria X. Dado que F(X) está acotada entre 0 y 1, podemos generar valores en U(0, 1) y sobre ellos aplicar $F^{-1}(X)$.

Ejemplo: Distribución exponencial

- La f.d.p y la Función de distribución acumulativa son:

$$f(x) = \lambda \cdot e^{-\lambda \cdot x}$$
$$F(x) = 1 - e^{-\lambda \cdot x}$$

$$F^{-1} = -\frac{1}{\lambda} ln(1-u)$$

- El algoritmo para generar valores aleatorios para esta distribución sería :

```
def exponential(media):
u = random(0,1)
return -log(1-u)/media
```

Ejemplo: Distribución discreta

- El método de la inversa es especialmente útil para generar valores de variables aleatorias discretas.
- **Ejemplo**

  Supongamos que tenemos una variable aleatoria discreta con espacio de muestreo S ={1, 2, 3}, y cuya funci´on de densidad es la siguiente:

  $$p(1) = 0{,}2 \quad p(2) = 0{,}3 \quad p(3) = 0{,}5$$

  Entonces su función de distribución sería:

  $$F(1) = 0{,}2 \quad F(2) = 0{,}2 + 0{,}3 = 0{,}5 \quad F(3) = 0{,}5 + 0{,}5 = 1$$

  Así, en el eje de F(x) podemos identificar tres intervalos: (0, 0.2], (0.2, 0.5] y (0.5, 1].
- Para obtener valores para esta variable, generaremos un valor para U(0, 1), y determinaremos su inversa a partir del intervalo en el que caiga. Si éste cae en el primer intervalo, el valor de F^{-1} será 1, si en el segundo 2, y si en el tercero 3.

Ejemplo: Distribución discreta

- Así, el algoritmo para generar valores para esta variable aleatoria sería el siguiente:

```
def genera_discreta(p1, p2, p3):
u= random(0,1)
if u<=p1:
        finv = 1
elif u<=p1+p2:
        finv = 2
else:
        finv = 3
return finv
```

- Recuerde que este tipo de distribuciones suele aparecer en los diagramas de sucesos, concretamente cuando un suceso planifica la ocurrencia de otros sucesos con una determinada probabilidad

Método del rechazo

- Cuando no podamos calcular la inversa de la función de distribución, lo que sucede en la mayor parte de las distribuciones continuas, entonces tendremos que basarnos en la función de densidad f(x).
- El método del rechazo utiliza una función de densidad g(x) para la cual sabemos generar números aleatorios (ej. una uniforme), y que además recubre por completo la función de densidad que queremos obtener, es decir:

$$a \cdot g(x) \quad f(x)$$

Método del rechazo

- El algoritmo para generar valores a partir de g(x) sería el siguiente:

```
repetir
    genera un valor x con densidad g
    genera un valor u con U(0, 1)
hasta que u·a·g(x) <= f(x)
devuelve x
```

- Para que funcione eficientemente, el algoritmo debe utilizar una función g(x) sencilla, y que se ajuste lo más posible a la función f(x). De ella dependerá el número de iteraciones del algoritmo.

Ejemplo

- Veamos un ejemplo presentado en [Raj Jain, 1991]. Supongamos que queremos generar una secuencia de valores para la distribución beta(2, 4), cuya función de densidad es:

$$f(x) = 20 \cdot x(1 - x)3$$

- Esta función puede ser recubierta por un rectángulo de altura 2,11, que es el máximo de la función. Por lo tanto, tomando g(x) = U(0, 1) y a = 2,11, obtendríamos el siguiente algoritmo:

```
def beta2_4():
    a = 2.11
    x = random(0,1)
    u = random(0,1)
    while (u*a <= (20*x*(1-x)**3)):
    x = random(0,1)
    u = random(0,1)
return x
```

Método de Convolución

- Si X es la suma de dos variables aleatorias Y_1 y Y_2, entonces la función de distribución de X puede obtenerse analíticamente mediante la convolución de las funciones de distribución de Y_1 y Y_2.
- Este es el principio del método de convolución: para obtener los valores de X bastará con generar 2 valores para Y_1 y Y_2, y devolver su suma.
- Es importante destacar que la suma de un conjunto de variables aleatorias (convolución) es un concepto diferente al de la suma de sus funciones de distribución (composición).

Método de Convolución

- Existen varias distribuciones que pueden generarse por convolución:
  - Una variable con distribución normal puede obtenerse sumando un número suficiente de variables aleatorias con cualquier distribución.
  - Una variable con distribución chi-cuadrado con k grados de libertad puede obtenerse sumando los cuadrados de k variables con distribución N(0, 1).
  - Una variable con distribución Erlang-k puede obtenerse sumando k variables con distribución exponencial.
  - La suma de dos variables con distribución U(0, 1) forma una variable con distribución triangular.