Facultad de Informática
Universidad Politécnica de Madrid

MÉTODOS DE SIMULACIÓN

Capítulo 2. Generación de números aleatorios

Departamento de Inteligencia Artificial

ÍNDICE

1. Introducción

Disponibilidad de un buen generador de números aleatorios.

Es un elemento esencial en muchas otras áreas de la Informática (algoritmos aleatorizados, verificación de algoritmos, complejidad de algoritmos, criptografía,...), de la Estadística (métodos de muestreo y remuestreo, contrastes Montecarlo, Inferencia Bayesiana...), ...

Además, un elemento importante en aplicaciones como juegos para ordenador, protectores de pantallas,...

La disponibilidad de generadores de números aleatorios en muchos entornos y compiladores haría pensar que para un mero usuario de la Simulación no sería necesario estudiar estas cuestiones.

Sin embargo, una lección que se deriva del estudio de algunos generadores comerciales es que **debemos actuar con sumo cuidado con ellos** (área de investigación bastante activa).

Problema: escoger una fuente de números aleatorios y obtener de esta fuente suficientes números para nuestro experimento de simulación.

Orígenes históricos.

Tablas de números aleatorios:

- Tippet (1927): Universidad de Cambridge, 10.000 números aleatorios de 4 dígitos basados en censos.
- Royo y Ferrer (1954): 250.000 resultados de la lotería nacional (INE).
- Rand Corporation (1954): 1 millón de números aleatorios mediante el uso de mecanismos físicos (ruleta electrónica, medición de ruido electrónico en circuitos...).

Inconveniente del uso de mecanismos físicos → falta de reproducibilidad.

Procedimientos algorítmicos de generación de números → La idea (von Neumann) es producir números que parezcan aleatorios, empleando las operaciones aritméticas del ordenador: partiendo de una semilla inicial ($u_0, u_1, \ldots, u_{-p+1}$), generar una sucesión mediante $u_i = d(u_{i-1}, \ldots, u_{i-p})$, para cierta función d.

¿qué se considera como secuencia de números aleatorios?

Definición clásica de Kolmogorov (Kolmogorov y Uspenskii, 1987) (asociada a la idea de complejidad algorítmica). Una sucesión de números es aleatoria si no puede producirse eficientemente mediante un programa más corto que la propia serie.

Definición basada en la Estadística. Una sucesión de números aleatorios (u_i) es una sucesión de números en (0,1) con las propiedades de

- Uniformidad en (0,1) y
- Aleatoriedad o independencia estadística.

Otras propiedades relativas a la **eficiencia computacional**:

1) Rapidez;
2) Poco consumo de memoria;
3) Portabilidad;
4) Sencillez en la implementación;
5) Reproducibilidad y mutabilidad;
6) Periodo suficientemente largo.

2. Contrastes empíricos

Bondad de ajuste o uniformidad

Contraste χ^2

El contraste χ^2 de Pearson es el más antiguo y válido para distribuciones continuas y discretas. Sin embargo, es poco potente, por lo que permite justificar el rechazo de una hipótesis, pero proporciona escaso soporte para su aceptación.

Supongamos que tenemos una muestra $x_1,\ldots,x_n$ ($n \geq 25$) de una población con función de distribución $F_n(x)$ desconocida y deseamos contrastar la hipótesis

$$H_0: F_n(x) = F_0(x),$$

para todo $x \in \Re$, donde $F_0(\mathrm{x})$ está completamente especificada (conocemos la distribución y los parámetros de la misma), frente a la alternativa

$$H_1: F_n(x) \neq F_0(x) \text{ para algún } x.$$

En nuestro caso, $F_0(x) = \mathrm{U}(0,1)$.

1. Agrupamos los n datos en k clases mutuamente excluyentes, $k \geq 5$, que cubran todo el rango posible de valores, siendo O_i a la frecuencia observada en la clase i.

2. Calculamos la frecuencia esperada, E_i, de la clase i de acuerdo con el modelo $F_0(x)$. Conociendo la probabilidad que asigna el modelo a la clase i tenemos

$$E_i = n \times p_i$$

3. Calculamos la discrepancia entre las frecuencias observadas y las esperadas mediante el modelo $F_0(x)$:

$$X^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

que se distribuye aproximadamente como una χ^2 cuando el modelo es correcto.

4. Determinamos los grados de libertad:
   - Si el modelo especifica las probabilidades p_i antes de tomar la muestra, entonces el nº de grados de libertad es $k - 1$.
   - Si las p_i se han calculado estimando r parámetros del modelo de máxima verosimilitud, entonces el nº de grados de libertad es $k - r - 1$.

5. Rechazamos el modelo cuando $X^2 \geq \chi^2_\alpha(k - r - 1)$ para un nivel de significación pequeño (0.05, 0.01, 0.001).

Capítulo 2. Generación de números aleatorios

En nuestro caso, para contrastar la uniformidad escogeremos k subintervalos de [0,1] de la misma longitud, siendo $p_i = 1/k$ y, por lo tanto, $E_i = n/k$ y $r = 0$, ya que no ha sido necesario estimar ningún parámetro de la distribución para obtener p_i.

Contraste Kolmogorov-Smirnov (K-S)

Es válido para distribuciones continuas, pero muy potente. Deseamos contrastar la hipótesis

$$H_0\text{: } F_n(\mathrm{x}) = F_0(\mathrm{x})$$

1. Ordenamos los valores muestrales de manera que $x_{(1)} \leq x_{(2)} \leq \ldots \leq x_{(n)}$

2. Calculamos la función de distribución empírica de la muestra:

$$F_n(\mathrm{x}) = \begin{cases} 0 & \text{si } x < x_{(1)} \\ r/n & \text{si } x_{(r)} \leq x < x_{(r+1)} \\ 1 & \text{si } x > x_{(r)} \end{cases}$$

3. Calculamos la máxima discrepancia entre la función de distribución empírica y la teórica contrastada. Estadístico bilateral de K-S

$$D_n = \sup_{x\ R} |F_n(\mathrm{x}) - F_0(\mathrm{x})|.$$

La distribución exacta de D_n está tabulada para valores seleccionados de $n \leq 40$ y del nivel de significación α.

Para muestras grandes, se utiliza la distribución asintótica de D_n, que viene dada, para todo $z \geq 0$, por

$$\lim_{n\to\infty} \mathrm{P}((n)^{1/2} D_n \leq z) = L(z)$$

donde $L(z)$ está tabulada y se comprueba que la aproximación es suficientemente buena para $n \geq 35$. Intuitivamente, esperamos que D_n sea pequeño cuando la hipótesis nula es cierta.

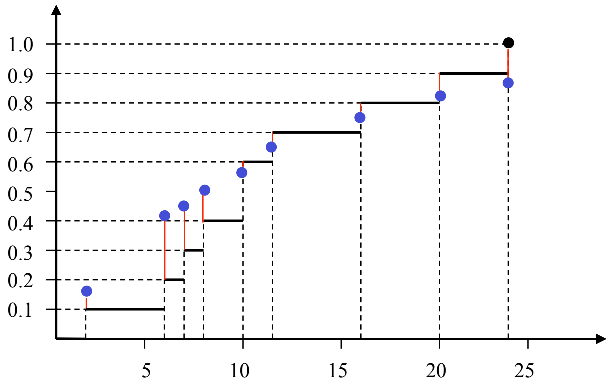
En nuestro caso particular de aleatoriedad, si $x_{(1)} \leq x_{(2)} \leq \ldots \leq x_{(n)}$ designa al estadístico de orden, $F_0(x_{(\mathrm{i})}) = x_{(\mathrm{i})}$, y como $F_n(x_{(\mathrm{i})}) = i/n$, resulta

$$D_n = \max_{1 \leq i \leq n} \left\{ \max \left[\left| \frac{i}{n} - x_{(i)} \right|, \left| x_{(i)} - \frac{i-1}{n} \right| \right] \right\}$$

Ejemplo. Contrastar si la siguiente muestra de duraciones de vida puede suponerse exponencial: 16, 8, 10, 12, 6, 10, 20, 7, 2, 24.

Ordenamos la muestra: 2, 6, 7, 8, 10, 10, 12, 16, 20, 24.

Representamos la función empírica:



x	$F_n(x)$	$F_0(x)$	$D_n(x)$
2	0.1	0.16	0.06
6	0.2	0.41	$\boxed{0.21}$
7	0.3	0.46	0.16
8	0.4	0.5	0.1
10	0.6	0.58	0.02
12	0.7	0.65	0.05
16	0.8	0.75	0.05
20	0.9	0.82	0.08
24	1	0.88	0.12

Función de distribución exponencial

$$F_0(x) = 1 - e^{-\lambda x}$$

Estimamos el parámetro a partir de la media muestral:

$$\frac{16+8+12+\ldots+24}{10} = 11.5 = 1/\lambda$$

$D_n = 0.21 < D(\alpha=0.2, n=10) = 0.322$

Aceptamos la hipótesis

Repetición de contrastes

Dada una secuencia de números, para contrastar su uniformidad formamos grupos de al menos 40 elementos.

A continuación, para cada uno de los conjuntos formados se calcula el nivel de discrepancia utilizado en el contraste χ^2.

Finalmente, contrastamos (aplicando, por ejemplo, Kolmogorov-Smirnov) que estos valores provienen de una distribución χ^2.

$$\underbrace{\underbrace{x_1, x_2, \ldots, x_{40}}_{\chi^2_1}, \underbrace{x_{41}, \ldots, x_{80}}_{\chi^2_2}, \underbrace{x_{81}, \ldots, x_{120}}_{\chi^2_3}, \underbrace{x_{121}, \ldots, x_{160}}_{\chi^2_4}, \underbrace{x_{161}, \ldots, x_{200}}_{\chi^2_5}, \underbrace{x_{201}, \ldots, x_{240}}_{\chi^2_6}, \underbrace{x_{241}, \ldots}_{\ldots}\ \ldots}_{\text{¿provienen de una distribución } \chi^2\text{?}}$$

Aleatoriedad o Independencia Estadística

Contraste de Rachas

Dada una secuencia de números $x_1, x_2, \ldots, x_n$.

1. Construimos una sucesión de símbolos binarios, asignando un 1 si $x_i \leq x_{i+1}$ y 0 si $x_i > x_{i+1}$.

2. Definimos como *racha creciente* (*decreciente*) de longitud l a un grupo de l unos (0's) consecutivos. Contabilizamos el número de rachas, que denotaremos como n_r.

   Puede comprobarse que el número total de rachas en una muestra de n observaciones independientes sigue una distribución aproximadamente normal (si $n \geq 40$) con parámetros

$$\mu = \frac{2n-1}{3} \qquad \sigma^2 = \frac{16n-29}{90}$$

3. Comprobamos si el número total de rachas observadas puede provenir con una probabilidad razonable de dicha distribución

$$¿P\left(\left|z_{N\left(\frac{2n-1}{3},\frac{16n-29}{90}\right)}\right| \geq n_r\right)?$$

   Los valores críticos vienen tabulados.

   Rechazaremos la hipótesis de independencia cuando el número de rachas sea significativamente grande o pequeño (alta dependencia positiva o negativa).

Ejemplo. Contrastar la aleatoriedad de la siguiente secuencia de números: 0.2, 0.4, 0.1, 0.3, 0.6, 0.7, 0.5, 0.8, 0.3, 0.2.

Construimos la secuencia de símbolos binarios para contabilizar las rachas:

0.2 0.4 0.1 0.3 0.6 0.7 0.5 0.8 0.3 0.2 $(n = 10)$

$$\underbrace{1}_{\text{crec}}\ \underbrace{0}_{\text{decr}}\ \underbrace{1\quad 1\quad 1}_{\text{crec}}\ \underbrace{0}_{\text{decr}}\ \underbrace{1}_{\text{crec}}\ \underbrace{0\quad 0}_{\text{decr}}$$

El número total de rachas es $n_r = 6$. La distribución del número de rachas en una muestra de tamaño 10 debería ser Normal(6.3, 1.45).

$$P\left(\left|z_{N(6.3,1.45)}\right| \geq 6\right) = P\left(z_{N(6.3,1.45)} \leq -6\right) + P\left(z_{N(6.3,1.45)} \geq 6\right) =$$

$$P\left(z_{N(0,1)} \leq \frac{-6-6.3}{\sqrt{1.45}}\right) + P\left(z_{N(0,1)} \geq \frac{6-6.3}{\sqrt{1.45}}\right) =$$

$$0 + P\left(z_{N(0,1)} \geq -0.25\right) = 1 - P\left(z_{N(0,1)} \leq -0.25\right) =$$

$$1 - 0.4030 = 0.5987$$

La probabilidad teórica de que el número de rachas en la secuencia sea mayor que 6 es 0.5987, por lo que rechazamos la hipótesis de aleatoriedad.

Contraste de rachas por encima y por debajo de la mediana

Otro contraste para calcular rachas se obtiene del recuento de observaciones que se sitúan por encima o por debajo de la mediana (en nuestro caso, la mediana será 0.5). Dada una secuencia de números $x_1, x_2, \ldots, x_n$.

1. Construimos una sucesión de símbolos binarios, asignando un 1 si $x_i \leq$ *mediana* y 0 si $x_i >$ *mediana.*

2. Sea k el número de unos en la sucesión (que será por hipótesis igual al número de ceros) e igual a $(n$-1)/2 si el número de observaciones es impar y no hay observaciones repetidas. Contabilizamos el número de rachas, n_r.

   En este caso, el número total de rachas en una muestra de n observaciones independientes sigue una distribución aproximadamente normal (si $n \geq 40$) con parámetros

$$\mu = k + 1 \qquad \sigma^2 = \frac{k(k-1)}{2k-1}$$

Test de Póker

Dada una sucesión de números aleatorios en [0,1], x_1, x_2,..., x_n:

1. Consideramos la conversión a los números enteros 1, 2,..., 10 mediante

reemplazar x_j	por	1	si	$0.0 \leq x_j < 0.1$
reemplazar x_j	por	2	si	$0.1 \leq x_j < 0.2$
…	…			…
reemplazar x_j	por	10	si	$0.9 \leq x_j < 1.0$

de modo que el nuevo cjto lo será de números enteros aleatorios entre 1 y 10.

2. Tomar los conjuntos formados por grupos de 5 nºs enteros sucesivos $\{x_{5j}, x_{5j+1}, ..., x_{5j+4}\}$ y determinar para cada uno cuál de los siguientes resultados se cumple:

- AAAAA → P(AAAAA) = 0.0001
- AAAAB → P(AAAAB) = 0.0045

- AAABB → P(AAABB) = 0.0090
- AAABC → P(AAABC) = 0.0720
- AABBC → P(AABBC) = 0.1080
- AABCD → P(AABCD) = 0.5040
- ABCDE → P(ABCDE) = 0.3024

El test de póker finaliza con el estudio de la bondad de ajuste a la anterior distribución basándose para ello en el test de la χ^2, que utilizará el recuento de estas 7 categorías o particiones para el conjunto generado.

Debido a lo baja que es la probabilidad de la categoría AAAAA, se suelen juntar las categorías AAAAA y AAAAB.

Si no fuera así, ya que el número esperado de ocurrencias de cada categoría debe ser al menos 5, se tendría que para la categoría AAAAA debería ser

$$(0.0001)(n/5) \geq 5,$$

es decir, contrastar $n \geq 250.000$ números aleatorios, lo que significaría un tamaño demasiado grande.

Ejemplo. Dada la siguiente muestra de 1500 números, realizar el contraste de Póker de aleatoriedad:

$$\begin{array}{cccccccccccccccccccc} 0.12 & 0.23 & 0.78 & 0.84 & 0.16 & 0.22 & 0.29 & 0.47 & 0.95 & 0.13 & 0.33 & 0.27 & 0.42 & 0.32 & 0.69 & 0.72 & 0.17 & 0.22 & 0.09 & 0.81 \; \ldots \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 2 & 3 & 8 & 9 & 2 & 3 & 3 & 5 & 10 & 2 & 4 & 3 & 5 & 4 & 7 & 8 & 2 & 3 & 1 & 9 \; \ldots \end{array}$$

$$\underbrace{2\;3\;8\;9\;2}_{AABCD} \quad \underbrace{3\;3\;5\;10\;2}_{AABCD} \quad \underbrace{4\;3\;5\;4\;7}_{AABCD} \quad \underbrace{8\;2\;3\;1\;9}_{ABCDE} \quad \ldots$$

Clases	Frec. observadas	Frec. Esperadas
AAAAA	3	$n \times p_{AAAAA} = 300 \times 0.0001 = 0.3$
AAAAB	5	$n \times p_{AAAAB} = 300 \times 0.0045 = 1.35$
AAABB	5	$n \times p_{AAABB} = 300 \times 0.0090 = 2.7$
AAABC	25	$n \times p_{AAABC} = 300 \times 0.0720 = 21.6$
AABBC	27	$n \times p_{AABBC} = 300 \times 0.1080 = 32.4$
AABCD	170	$n \times p_{AABCD} = 300 \times 0.5040 = 151.2$
ABCDE	65	$n \times p_{ABCDE} = 300 \times 0.3024 = 90.72$

$$X^2 = \frac{(8-1.65)^2}{1.65} + \frac{(5-2.7)^2}{2.7} + \frac{(25-21.6)^2}{21.6} + .. + \frac{(65-90.72)^2}{90.72} = 24.43 + 1.959 + 0.5351 + \ldots + 7.314 = 37.46$$

El número de grados de libertad es k-r-1 = 6-0-1 = 5 y de la tabla de la distribución de la χ^2 tenemos $\chi^2_{0.995}(5) = 16.7 < X^2 = 37.46$, por lo tanto, **aceptamos la hipótesis**.

Una versión más simple de este contraste que facilita una programación más sencilla, Knuth (1998), consiste en encontrar para cada conjunto de 5 enteros su número de valores distintos.

Así, tendríamos 5 categorías:

- 5 valores: todos diferentes
- 4 valores: un par
- 3 valores: dos pares o tres iguales
- 2 valores; cuatro iguales
- 1 valor: los cinco iguales

Esta clasificación es más sencilla de obtener de manera sistemática, siendo el contraste con calidad próxima al de póker.

Otros contrastes que se pueden utilizar para la aleatoriedad son el contraste de permutaciones, el contraste de huecos y el contraste de Box-Pierce.

ÍNDICE

3. Generadores congruenciales

Los generadores más populares son probablemente los congruenciales, debidos a Lehmer (1951). Siguen la fórmula de recursión:

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

que proporciona números enteros en $[0, m)$, para un multiplicador a, sesgo b, módulo m y semilla x_0, donde suponemos que $a, b \in \{0, 1, ..., m\text{-}1\}$. La utilización de la misma semilla llevará a la misma secuencia de nºs aleatorios (reproducibilidad y mutabilidad).

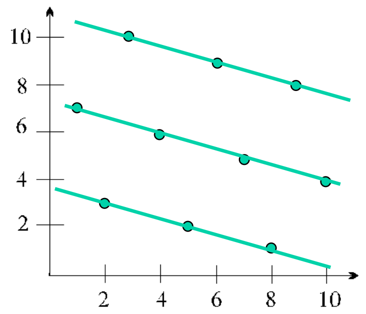
Se convierten en números uniformes en $[0,1)$ dividiendo por m, $\quad u_n = x_n / m$.

Si $b = 0$, se denominan multiplicativos: $\quad x_{n+1} = a\, x_n \bmod m$,

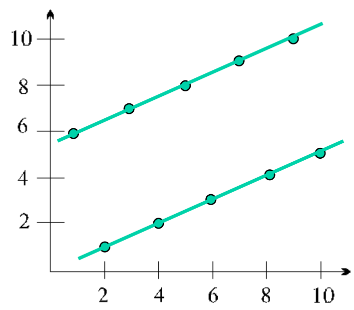
A pesar de su aparente simplicidad y previsibilidad, una selección cuidadosa de los parámetros (a, b, m) permite obtener de manera eficiente sucesiones de números suficientemente largas y aleatorias para muchos propósitos.

Consideremos los generadores multiplicativos (b = 0):

Caso	a	m	x_0	Salidas											
1	6	13	1	6	10	8	9	2	12	7	3	5	4	11	$\boxed{1}$
2	7	13	10	5	9	11	12	6	3	8	4	2	1	7	$\boxed{10}$
3	5	13	5	12	8	1	$\boxed{5}$	12	8	1	5	12	8	1	5
4	7	11	5	2	3	10	4	6	9	8	1	7	$\boxed{5}$	2	3
5	6	11	3	7	9	10	5	8	4	2	1	6	$\boxed{3}$	7	9



caso 4



caso 5

Algunas observaciones son:

1) Un generador congruencial tiene ciclos.

2) La longitud del ciclo depende de la selección de los parámetros (comparar casos 1 y 3).

3) Dentro de selecciones de parámetros que conducen a una misma longitud de ciclo, algunas salidas parecen más aleatorias que otras (comparar casos 1 y 2).

4) La representación de los pares (x_i, x_{i+1}) sugiere que éstos se disponen en un número finito de rectas (representación de tuplas → hiperplanos).

Trivialmente, el periodo máximo m se alcanza para $a = b = 1$.

Además, cuando $b = 0$ (generador multiplicativo) el máximo periodo es m-1.

El periodo depende de la selección de a y b, para lo que se dispone de algunos resultados que indicamos aquí:

Proposición: Un generador congruencial tiene periodo máximo m si y sólo si:
1) mcd $(b, m) = 1$;
2) $a = 1 \bmod p$ para cada factor primo p de m;
3) $a = 1 \bmod 4$ si 4 divide a m.

Puesto que b tiene en la práctica el efecto de una traslación, estamos interesados en estudiar resultados para generadores multiplicativos.

El valor de m se escoge para que la operación Y mod m se realice de forma sencilla. Para un ordenador binario, si $m = 2^\beta$, lo único que debemos hacer para ejecutar tal operación es retener los últimos β bits de Y. Entonces:

Proposición. Un generador multiplicativo con módulo $m = 2^\beta \geq 16$ tiene periodo máximo $m/4$ si y sólo si $a \bmod 8 = 3$ ó $a \bmod 8 = 5$ y x_0 es impar.

También hay procedimientos sencillos para el caso $m = 2^\beta+1$ (Ripley, 1987).

Proposición. Un generador multiplicativo tiene periodo m-1 sólo si m es primo. El periodo divide a m-1 y es m-1 si y sólo si a es una raíz primitiva de m-1, es decir, $a \neq 0$, $a^{(m-1)/p} \neq 1 \bmod m$, para todos los factores primos p de m-1.

Dos elecciones frecuentes son $m = 2^{31}-1$, en ordenadores de 32 bits, y $m = 2^{16}+1$, en ordenadores de 16 bits.

Mencionemos que se conocen condiciones computables para determinar raíces primitivas.

De hecho, una vez encontrada una, el resto proviene de:

Proposición. Si a es una raíz primitiva de m, $a^k \bmod m$ lo es siempre que mcd$(k, m-1) = 1$.

Entonces, podemos elegir m para que la implementación del generador sea eficiente y, una vez elegido m, podemos elegir a y b en su caso, para que el generador sea de ciclo máximo.

Debemos aún estudiar con más detalle tal elección, pues tenemos muchos grados de libertad. Para motivar la elección, apelamos a nuestra cuarta observación: un resultado de Marsaglia (1968) demuestra que "sucesiones solapantes de n números de un generador multiplicativo caen en, a lo sumo, $(n!m)^{1/n}$ hiperplanos paralelos". (Estructura reticular de un generador)

Por ejemplo, tenemos las siguientes cotas en un par de casos representativos

	$n = 3$	$n = 5$	$n = 7$	$n = 9$	$n = 10$
$m = 2^{16}$	73	23	16	14	13
$m = 2^{32}$	2953	220	80	48	41

Los ejemplos que utilizamos como introducción muestran lo pésima que puede ser la estructura reticular de estos generadores.

Un ejemplo famoso es el de la rutina RANDU, que IBM proporcionaba a mediados de los 70, con $m = 2^{31}$, $a = 65539$ y $b = 0$. ¡Para este generador, las tripletas consecutivas de números caen en 15 planos!

Fishman y Moore (1986) utilizaron el contraste espectral para seleccionar buenos multiplicadores (entre las 23093 raíces primitivas de m).
Criterio: si para $2 \leq k \leq 6$ y cada conjunto de hiperplanos paralelos, la distancia euclídea entre hiperplanos adyacentes no supera la mínima distancia alcanzable en más del 25%.

Identificaron así 410 multiplicadores.

Fishman realizó una búsqueda exhaustiva de los 23093 anteriores con un criterio del 30%, encontrando varios buenos multiplicadores, entre ellos 16807, aunque, por ejemplo, $a = 48271$ y $a = 69621$ parecen ser mejores. Sin embargo, dada la mayor experiencia con $a = 16807$ aún se suele proponer tal multiplicador como el estándar.

Para una discusión más detallada véase Park y Miller (1988) y Press *et al* (1992).

Generador congruencial estándar

Arquitecturas de 32 bits, Park y Miller (1988) → mínimo estándar:
1) Ser de periodo máximo;
2) Que su salida parezca aleatoria;
3) Que se pueda implementar de forma eficiente en aritmética de 32 bits.

Por ser primo es $m = 2^{31}-1$. El multiplicador que se ha convertido en estándar es el asociado a $a = 7^5 = 16807$. 7 es una raíz primitiva de $2^{31}-1$.

```
function aleator (isemilla)
  isemilla= mod (isemilla × 16807.d0, 2147483647.d0)
  aleator = isemilla / 2147483647
  return
end
```

Puede dar problemas de desbordamiento pues *isemilla* × 16807.d0 puede valer hasta 1.03×2^{45}. Una forma de comprobar el correcto funcionamiento del generador consiste en comenzar con una *isemilla* inicial igual a 1 y comprobar que después de 10.000 llamadas *isemilla* toma el valor 1043618065.

Método de Schrage (Bratley, Fox y Schrage, 1983). Su idea básica es realizar la operación de módulo antes que el producto.

Supongamos que escribimos $m = aq + r$
con
$$q = m \text{ div } a \quad y \quad r = m \bmod a$$

Entonces tenemos
$$f(z) = az \bmod m = az - m\,(az \text{ div } m) + m\,(z \text{ div } q) - m\,(z \text{ div } q)$$
$$= \gamma(z) + m\,\delta(z)$$
con
$$\gamma(z) = a\,(z \bmod q) - r\,(z \text{ div } q)$$
$$\delta(z) = (z \text{ div } q) - (az \text{ div } m)$$

Se puede demostrar que si $r < q$ para todo $z \in \{0,1,\ldots,m-1\}$ resulta
1) $\gamma(z) \in \{0,1\}$
2) $a\,(z \bmod q)$, $r\,(z \text{ div } q) \in \{0,1,\ldots,m-1\}$
3) $|\gamma(z)| \leq m-1$

Así, puesto que $1 \leq f(z) \leq m$-1, resulta

$$\gamma(z) = 0 \text{ si y solo si } \quad 1 \leq \gamma(z) \leq m\text{-}1$$
$$\gamma(z) = 1 \text{ si y solo si } \quad -(\text{m-1}) \leq \gamma(z) \leq \text{-}1$$

Por tanto, para evaluar $f(z)$, evaluamos $\gamma(z)$. Si $\gamma(z) > 0$, hacemos $f(z) = \gamma(z)$; en caso contrario $f(z) = \gamma(z) + m$.

Por ejemplo, para a = 16807 y m = 2147483647, tenemos q = 127773 y r = 2863, y la implementación en FORTRAN queda de la siguiente forma:

```
function aleato2 (isemilla)
  k = isemilla/127773
  isemilla = 16807 ×(isemilla - k × 127773) - 2836 × k
  if (isemilla.lt0)    isemilla = isemilla + 2147483647
  aleato2 = 1./2147483647 × isemilla
  return
end
```


A pesar de estas buenas propiedades, se han diseñado contrastes muy potentes que permiten rechazar la aleatoriedad de los generadores congruenciales (L'Ecuyer, 1994).

Tales contrastes explotan la estructura reticular de estos generadores.

Por ello y por la necesidad de disponer de generadores de periodo más largo, describimos algunos otros intentos de definir buenos generadores.

ÍNDICE

Capítulo 2. Generación de números aleatorios

4. Otros generadores

4.1. Generadores de Registro de Desplazamiento

Los generadores congruenciales pueden generalizarse a recursiones lineales de orden mayor. Para $k \geq 1$, m primo

$$x_n = (a_1 x_{n-1} + \ldots + a_k x_{n-k}) \bmod m$$

y el generador se denomina recursivo múltiple. (L'ecuyer, 1994)

Su estudio se asocia al estudio del polinomio característico

$$P(z) = z^k - a_1 z^{k-1} - \ldots - a_k$$

sobre el álgebra finita F_m con m elementos.

Cuando el polinomio es primitivo, Niederreiter (1992), el periodo del generador es m^k-1. El estudio de tales polinomios y la implementación del generador cuando m es grande no es sencillo, por lo que se escoge m pequeño, habitualmente 2, dado que permite una sencilla implementación.

Parece ser que estos generadores tienen buenas propiedades de aleatoriedad, sin embargo, producen estructuras reticulares, como los congruenciales, lo que ha llevado a una cierta polémica y disuasión sobre su calidad.

Ejemplo. Tomando $m = 2$ tenemos el generador $x_n = (a_1 x_{n-1} + \ldots + a_k x_{n-k}) \bmod 2$, donde $a_1,\ldots,a_k \in \{0,1\}$, que es equivalente a reescribir

$$x_n = x_{n-j1} \text{ XOR } x_{n-j2} \text{ XOR } \ldots \text{ XOR } x_{n-jk'}$$

cuando $a_{n-j1} = a_{n-j2} = \ldots = a_{n-jk'} = 1$ y $a_j = 0$ para los otros j.

Una elección habitual es escoger polinomios de la forma $P(z) = z^k - z^{k-r} - 1$ o equivalentemente $x_n = x_{n-r} \text{ XOR } x_{n-k}$

Valores habituales para k y n (garantizan un periodo máximo):

$$\left\{ \begin{array}{l} k = 607 \quad r = 273 \\ x_n = x_{n-273} \text{ XOR } x_{n-607} \end{array} \right\} \text{ o } \left\{ \begin{array}{l} k = 521 \quad r = 32 \\ x_n = x_{n-32} \text{ XOR } x_{n-521} \end{array} \right\}$$

A partir de la sucesión de bits pueden generarse números en (0,1). La propuesta más utilizada son los denominados generadores de Tausworthe (1965):

Para un l y t dados, se define

$$u_i = \sum_{s=1}^{l} 2^{-s} b_{it+s}$$

Cuando $\text{mcd}(2^k-1, t)=1$ se denominan propios y tienen periodo 2^k-1. Estos generadores tienen buenas propiedades en el sentido de su aleatoriedad aunque producen estructuras reticulares similares a los de los generadores congruenciales.

Uno de los principales generadores actuales perteneciente a esta clase (de registro de desplazamiento) es el girador de Mersenne (Mersenne twister method, Matsumoto & Nishimura, 1998), que goza de buenas propiedades estadísticas.

4.2. Generadores de Fibonacci Retardados

Se trata de otra generalización de los generadores congruenciales y parten de la semilla inicial $(x_1, x_2,..., x_r)$ y usan la recursión $x_i = x_{i-r} \circ x_{i-s}$, donde r y s son retardos enteros que satisfacen $r > s$ y $\circ$ es una operación binaria que suele ser +, -, × o XOR.

Típicamente, los elementos iniciales son enteros y la operación binaria es la suma módulo 2^n.
La caracterización del periodo máximo de los generadores de Fibonacci retardados se basa en el análisis de sucesiones lineales recursivas de enteros de la forma

$$x_i = (a_1x_{i-1} + a_2x_{i-2} + \ldots + a_rx_{i-r}) \bmod m$$

Marsaglia (1985). Cuando $r = 17$, $s = 5$ y se escoge la operación + ó - sobre los enteros mod 2^n, el periodo es $(2^{17}-1)2^{n-1}$. Cuando es XOR, el periodo es $2^{17}-1$.

La implementación de un generador de este tipo es sencilla mediante una lista circular y dos punteros.

4.3. Generadores no lineales

Dada la estructura reticular de los generadores lineales, se sugiere utilizar generadores no lineales. Se distinguen dos formas de introducir no linealidad:

1. Usar un generador con función de transición lineal, produciendo la salida mediante una transformación no lineal del estado.

2. Usar un generador con función de transición no lineal.

No producen una estructura reticular como la de los lineales. Su estructura es altamente no lineal: típicamente, un hiperplano t-dimensional tendrá a lo sumo t t-uplas solapantes de números.

Niederreiter (1992) proporciona estudios teóricos sobre su estructura que sugiere sus buenas propiedades teóricas. Sin embargo, no existen aún implementaciones rápidas con parámetros bien contrastados, aunque el método que parece más prometedor, véase Eichenauer-Herrmann (1995), es el denominado método congruencial de inversión explícita.

Método congruencial de inversión explícita

Sea $m \geq 5$ un número primo y $F_m=\{0,1,\ldots,m\text{-}1\}$ el álgebra finita de orden m.

Para un entero z, se define $\overline{z} \in F_m$,

$$\overline{z} \equiv z^{m-2} \bmod m$$

que es la inversa de z para la multiplicación en F_m, si $\overline{z} \neq 0 \bmod m$.

Existen algunos algoritmos eficientes para su cálculo.

Dados $a, b \in F_m$, $a \neq 0$, la sucesión es

$$y_n = \overline{a\,n + b} \qquad n \geq 0$$

Su periodo máximo es m.

No hay tanta experiencia en los generadores no lineales como con los lineales.

Entre los no lineales ha emergido como principal opción el algoritmo Advance Encription Standard (AES), Hellekalek y Wegenkittl (2003).

4.4. Combinación de generadores

Para incrementar el periodo e intentar evitar las regularidades que muestran los generadores lineales congruenciales se ha sugerido combinar diferentes generadores para obtener uno híbrido, que sea de mayor calidad que los anteriores. Tales combinaciones pueden considerarse heurísticas, algunas con resultado bastante pobre.

El fundamento es esencialmente empírico, aunque también se han desarrollado algunos aspectos teóricos.

1. Se ha observado que el periodo de un generador híbrido es, en general, bastante más largo que el de sus componentes siendo, además, posible su determinación.

2. Hay resultados teóricos que sugieren que algunas formas de combinación de generadores mejoran su comportamiento estadístico. Por ejemplo, consideremos dos sucesiones aleatorias (x_n), (y_n) y la sucesión combinada elemento a elemento de ambas (z_n), donde $z_n = x_n \circ y_n$ y $\circ$ representa alguna operación binaria.

Ideas básicas para combinar generadores son el barajeo y composición.

Un análisis teórico de esta familia se ha llevado a cabo con esquemas bastante simples, por lo que no hay garantía de suficiente calidad, aunque algunos contrastes sugieren potencialidad de estas ideas.

Permutación fija (**barajeo**): Se generan los números en bloques de longitud L y se aplica una permutación fija a cada bloque antes de su uso.

Permutación aleatoria (**barajeo**): (MacLaren y Marsaglia, 1965): consiste en aplicar una permutación aleatoria a una sucesión. Si tenemos dos sucesiones aleatorias (u_n), (v_n) y $T(0),\ldots,T(k-1)$ están inicialmente ocupados por los valores $u_1,\ldots,u_k$, en cada etapa utilizamos v_n para seleccionar aleatoriamente un elemento de T: hacemos $J = \text{ent}\,(kv_{k+1})$, salimos con $T(J)$ y lo reemplazamos con u_{n+1}.

Ejemplo (permutación aleatoria, barajeo)

Partimos de dos sucesiones aleatorias:

$(u_n) \rightarrow$ 0.7 0.2 0.32 0.84 0.25 0.12 0.33 0.47 0.84 0.72 …

$(v_n) \rightarrow$ 0.1 0.7 0.23 0.42 0.35 0.21 0.47 0.72 0.68 0.12 …

Tomamos $k = 5$ y hacemos

$T(0) = u_1 = 0.7$ $T(1) = u_2 = 0.2$ $T(3) = 0.32$ $T(4) = 0.84$ $T(5) = 0.25$

Tomamos valores de la segunda secuencia aleatoria y construimos los índices que me señalarán los elemento de T que debemos devolver, a la vez que actualizo T:

v_1=0.1$\rightarrow j = \text{ent}(6\times0.1)=0.6 \rightarrow j = 0 \rightarrow$ el valor que debemos devolver es $T(0) =$ 0.7. Hago $T(0) = u_6 = 0.12$

v_2=0.7$\rightarrow j = \text{ent}(6\times0.7)=4.2 \rightarrow j = 4 \rightarrow$ el valor que debemos devolver es $T(4) =$ 0.84. Hago $T(1) = u_7 = 0.33$

….

Capítulo 2. Generación de números aleatorios

L'Ecuyer (1988) describe la **composición** de generadores congruenciales lineales multiplicativos de orden k = 1 con módulos primos distintos $m_1,\ldots,m_I$. Si x_{in} denota el estado del generador i-ésimo en la etapa n, se define la combinación $z_n = (\sum_{i=1}^{I} \alpha_i x_{in}) \bmod m_1$ para algunos enteros α_i, que suelen tomar la forma $(-1)^{i-1}$.

$$x^1_{n+1} = a\, x^1_n \bmod m_1 \qquad \cdots \qquad x^I_{n+1} = a\, x^I_n \bmod m_I$$

$$\Downarrow \qquad\qquad\qquad\qquad \Downarrow$$

$$\underbrace{x^1_{n+1} \qquad \cdots \qquad x^I_{n+1}}$$

$$z_{n+1} = (\textstyle\sum_{i=1}^{I} \alpha_i x^i_{n+1}) \bmod m_1$$

Algunos de los mejores generadores disponibles hoy en día corresponden a la composición de generadores múltiples recursivos, L'Ecuyer (2006), uno de los más empleados es el denominado MRG32k3a.

Otros generadores basados en la idea de **barajeo**:

- Método de Bays-Durham (1976)

Otros generadores basados en la idea de **composición**:

- Algoritmo Super-Duper-73 (Marsaglia et al., 1973): Combinación XOR de un generador lineal congruencial ($m=2^{32}$ y $a=69069$) con un pequeño generador de registro de desplazamiento.

- Algoritmo Super-Duper-64 (Marsaglia 2002): Generador 64 bits mediante una combinación aditiva de un generador lineal congruencial con un LFSR (linear feedback shift-register).

- Algoritmo KISS (Marsaglia, 1995): Composición de un generador lineal congruencial y uno de registro de desplazamiento.

- Generador de Wichmann-Hill (1982, 1984 corrections)

- Método MWC (Multiply With Carry) (Couture y L'Ecuyer, 1997)

4.5. Generadores Paralelos

La migración hacia arquitecturas paralelas van haciendo necesaria una adaptación y una revisión de los procedimientos actuales de generación de nºs aleatorios.

La forma más simple y obvia de generar números aleatorios con procesamiento en paralelo es utilizar un solo generador fuente.

La situación más sencilla se da cuando se considera un punto único de comienzo para ese generador, que proporcionará números a todos los procesadores que los necesiten. Hay dos inconvenientes:

1. A menos que haya una sincronización en la generación de los números aleatorios no será posible garantizar que todos los procesos particulares reciban siempre la misma secuencia exacta de números.
2. Para sistemas paralelos basados en "paso de mensajes" puede haber un aumento considerable de gasto en la transmisión de los números generados a los procesos que los necesitan.

Una forma sencilla de crear generadores separados para conjuntos de procesos consiste en utilizar puntos de comienzo predeterminados, es decir, utilizar un único generador común con un conjunto de semillas preseleccionadas para cada proceso.

La ventaja de este procedimiento es que todos los procesos usan el mismo código, aunque las sucesiones generadas y utilizadas por cada uno serán distintas.

4.6. Generadores Comerciales

Artículo de Park y Miller (1988) como advertencia sobre la mala calidad de algunos generadores comerciales.

IMSL implementa generadores multiplicativos de módulo $m = 2^{31}-1$ y multiplicadores $a = 16807$, 397204094 y 950706376.

El lenguaje de simulación SIMSCRIPT II.5 implementa el mismo tipo de generador con multiplicador $a = 630360016$, proporcionando semillas suficientemente separadas para producir sucesiones independientes.

El entorno estadístico S-PLUS implementa el algoritmo Super-Duper de Marsaglia basado en un generador multiplicativo y un generador de Tausworthe.

SPSS implementa el girador de Mersenne (Mersenne twister method).

La biblioteca estándar de Unix y Visual Basic utilizan el generador congruencial:

$$x_{n+1} = (1140671485 x_n + 12820163) \bmod 2^{24},$$

$$u_n = x_n / 2^{24}.$$

Java, en su clase java.util.Random, utiliza el mismo generador pero sale con

$$u_n = (2^{27} \lfloor x_{2n}/2^{22} \rfloor + \lfloor x_{2n+1}/2^{21} \rfloor)/2^{53}.$$

En Excel-97 se emplea el generador

$$u_{n+1} = (9821.0 u_n + 0.211327) \bmod 1,$$

mientras que en Excel-03 y Excel-07 se emplea el generador de Wichman-Hill.

Press *et al* (1992) incluyen, entre otros, el generador mínimo estándar, el generador mínimo estándar barajado según el método de Bays-Durham y el generador de L'Ecuyer.

Press *et al* (2007) se incluyen generadores para las aritméticas de 32 y 64 bits. En ambos casos se utiliza una combinación de los generadores: el método Xorshift (Marsaglia, 2003); el método MWC (Multiply With Carry, Couture y L'Ecuyer, 1997) con base $b = 2^{32}$; un generador congruencial lineal módulo 2^{64} y un generador congruencial lineal multiplicativo módulo 2^{64}.

Las combinaciones de estos generadores básicos se realizan mediante operaciones XOR, desplazamiento de bits a izquierda o derecha, la suma de valores obtenidos de diferentes generadores o la introducción de la secuencia de números aleatorios generada por uno de ellos como entrada en otro.

ÍNDICE

5.1 Contraste espectral

Se basa en "calcular la distancia máxima entre los hiperplanos adyacentes, tomándose el máximo sobre el conjunto de todos los hiperplanos paralelos que cubran los puntos". Cuanto mayor es la distancia, peor es el generador (Casos 4 y 5: En el primer caso, la máxima distancia es 3.479; en el segundo, es 4.919).

Para llevar a cabo el contraste, para k-úplas, hemos de resolver el problema de optimización

$$V_k = \min(\alpha_1{}^2+\alpha_2{}^2+\ldots+\alpha_k{}^2)^{1/2}$$
s.a
$$\alpha_1+a\alpha_2+\ldots+a^{k-1}\alpha_k = mq$$
$$(\alpha_1, \alpha_2,\ldots, \alpha_k, q)\in\mathbf{Z}$$

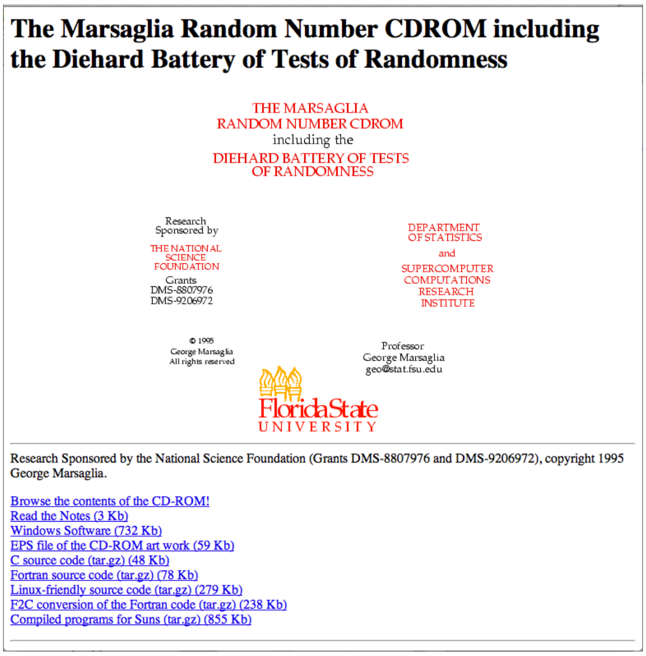
El valor del contraste espectral es m/V_k, véase Ripley (1987).

El problema es de programación entera no lineal, y para su resolución eficiente hemos de apelar a diversos heurísticos, como se expone en Knuth (1981). Resulta muy costoso para valores de k mayores que 6.

5.2 Diehard battery of test of Randomness

Marsaglia (1985)

www.stat.fsu.edu/pub/diehard



The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness

THE MARSAGLIA RANDOM NUMBER CDROM including the DIEHARD BATTERY OF TESTS OF RANDOMNESS

Research Sponsored by THE NATIONAL SCIENCE FOUNDATION Grants DMS-8807976 DMS-9206972

DEPARTMENT OF STATISTICS and SUPERCOMPUTER COMPUTATIONS RESEARCH INSTITUTE

© 1995 George Marsaglia All rights reserved

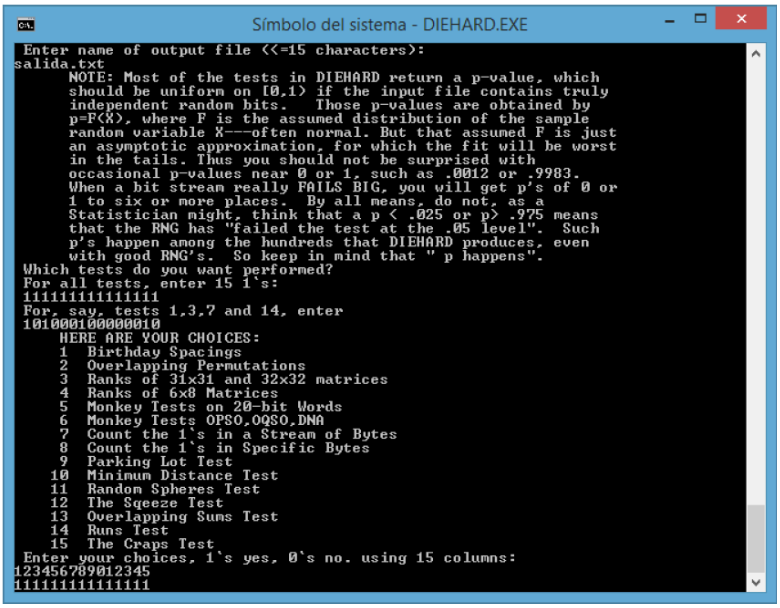
Professor George Marsaglia geo@stat.fsu.edu

FloridaState UNIVERSITY

Research Sponsored by the National Science Foundation (Grants DMS-8807976 and DMS-9206972), copyright 1995 George Marsaglia.

Browse the contents of the CD-ROM!
Read the Notes (3 Kb)
Windows Software (732 Kb)
EPS file of the CD-ROM art work (59 Kb)
C source code (tar.gz) (48 Kb)
Fortran source code (tar.gz) (78 Kb)
Linux-friendly source code (tar.gz) (279 Kb)
F2C conversion of the Fortran code (tar.gz) (238 Kb)
Compiled programs for Suns (tar.gz) (855 Kb)

Mediante consola de comandos, debemos indicar el fichero de entrada sobre el que se ejecutarán los 15 test y el de salida, en el que se guardarán los resultados de los mismos.

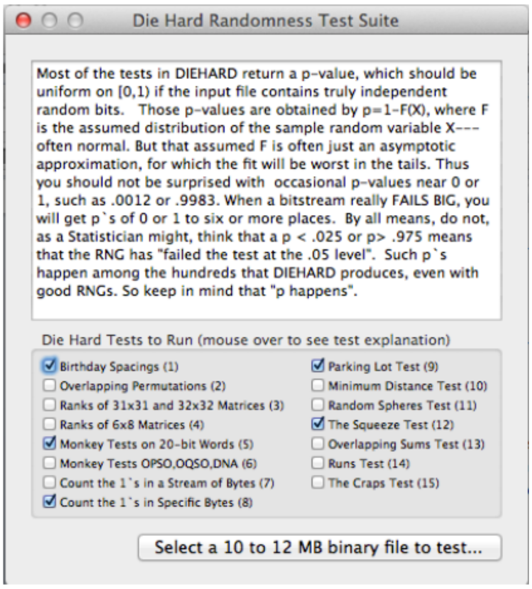


Símbolo del sistema - DIEHARD.EXE

```
 Enter name of output file (<=15 characters):
salida.txt
        NOTE: Most of the tests in DIEHARD return a p-value, which
        should be uniform on [0,1) if the input file contains truly
        independent random bits.   Those p-values are obtained by
        p=F(X), where F is the assumed distribution of the sample
        random variable X---often normal. But that assumed F is just
        an asymptotic approximation, for which the fit will be worst
        in the tails. Thus you should not be surprised with
        occasional p-values near 0 or 1, such as .0012 or .9983.
        When a bit stream really FAILS BIG, you will get p's of 0 or
        1 to six or more places.  By all means, do not, as a
        Statistician might, think that a p < .025 or p> .975 means
        that the RNG has "failed the test at the .05 level".  Such
        p's happen among the hundreds that DIEHARD produces, even
        with good RNG's.  So keep in mind that " p happens".
 Which tests do you want performed?
 For all tests, enter 15 1`s:
 111111111111111
 For, say, tests 1,3,7 and 14, enter
 101000100000010
      HERE ARE YOUR CHOICES:
         1  Birthday Spacings
         2  Overlapping Permutations
         3  Ranks of 31x31 and 32x32 matrices
         4  Ranks of 6x8 Matrices
         5  Monkey Tests on 20-bit Words
         6  Monkey Tests OPSO,OQSO,DNA
         7  Count the 1`s in a Stream of Bytes
         8  Count the 1`s in Specific Bytes
         9  Parking Lot Test
        10  Minimum Distance Test
        11  Random Spheres Test
        12  The Sqeeze Test
        13  Overlapping Sums Test
        14  Runs Test
        15  The Craps Test
 Enter your choices, 1`s yes, 0`s no. using 15 columns:
123456789012345
111111111111111
```

DIE HARD Randomness Test Suite

(para Mac, de CHS Systems, LLC)



Admite ficheros de un máximo de 12 MB

Capítulo 2. Generación de números aleatorios

Genera un fichero de datos plano con los resultados

```
|----------   Monkey Tests on 20-bit Words   (5) --------------|
|--------------------------------------------------------------|
|                  THE BITSTREAM TEST                          |
|The file under test is viewed as a stream of bits. Call them  |
|b1,b2,... .  Consider an alphabet with two "letters", 0 and 1 |
|and think of the stream of bits as a succession of 20-letter  |
|"words", overlapping.  Thus the first word is b1b2...b20, the |
|second is b2b3...b21, and so on.  The bitstream test counts   |
|the number of missing 20-letter (20-bit) words in a string of |
|2^21 overlapping 20-letter words.  There are 2^20 possible 20 |
|letter words.  For a truly random string of 2^21+19 bits, the |
|number of missing words j should be (very close to) normally  |
|distributed with mean 141,909 and sigma 428.  Thus            |
| (j-141909)/428 should be a standard normal variate (z score)|
|that leads to a uniform [0,1) p value.  The test is repeated  |
|twenty times.                                                 |
|--------------------------------------------------------------|

OVERLAPPING 20-TUPLES BITSTREAM  TEST
 (20 bits/word, 2097152 words 20 bitstreams. # of missing words
     should average 141909.33 with sigma=428.00.)
---------------------------------------------------------------
   BITSTREAM test results

Bitstream   # missing words z-score      p-value
    1          140994          -2.14       0.983767
    2          142371           1.08       0.140368
    3          141009          -2.10       0.982292
    4          142485           1.35       0.089309
    5          141433          -1.11       0.867129
    6          141752          -0.37       0.643412
    7          141609          -0.70       0.758569
    8          141690          -0.51       0.695833
    9          141439          -1.10       0.864095
   10          142432           1.22       0.111007
   11          141849          -0.14       0.556048
   12          141918           0.02       0.491919
   13          142066           0.37       0.357163
   14          142523           1.43       0.075813
   15          141456          -1.06       0.855242
   16          142025           0.27       0.393481
   17          141251          -1.54       0.937995
   18          142848           2.19       0.014148
   19          141531          -0.88       0.811638
   20          141978           0.16       0.436266
```

5.2 NIST (National Institute of Standards and Technology)

Rukhin et al. (2001)

http://csrc.nist.org/rng

1. Frequency (Monobit) Test
2. Frequency Test within a Block
3. Runs Test
4. Test for the Longest Run of Ones in a Block
5. Binary Matrix Rank Test
6. Discrete Fourier Transform (Spectral) Test
7. Non-overlapping Template Matching Test
8. Overlapping Template Matching Test
9. Maurer's "Universal Statistical" Test
10. Lempel-Ziv Compression Test
11. Linear Complexity Test
12. Serial Test
13. Approximate Entropy Test
14. Cumulative Sums (Cusum) Test
15. Random Excursions Test
16. Random Excursions Variant Test

Destinadas a ser usadas en **criptografía**; con énfasis en test de bits.

5.2 Tuftest

Marsaglia y Tsang (2002)

Artículo titulado: “Some difficult-to-pass tests of randomness”. Presentan tres pruebas Tuftest que según sus propias palabras son más exigentes que toda la batería Diehard:

- **Prueba del máximo común divisor** (MCD).

En el algoritmo de Euclides para encontrar el mcd de dos números enteros aparecen tres cantidades significativas:

- el número de iteraciones *k* que se requieren para calcular el mcd,
- una secuencia de longitud variable de enteros con los cocientes parciales,
- el mcd.

La prueba consiste en elegir pares de enteros al azar y construir tres listas (1) la del número de iteraciones *k* que son variables aleatorias independientes igualmente distribuidas (iid), la lista de los cocientes parciales cuyos valores no son iid, y (3) los mcd que sí son iid.

Marsaglia usa distribuciones empíricas de los valores *k* y de los mcd para comprobar la aleatoriedad de los números pseudo aleatorios producidos por generadores. La disparidad más frecuente sucede en la distribución de los k, el número de pasos para terminar el algoritmo de Euclides.

- **La prueba del cumpleaños**

Se basa que si se toman m cumpleaños aleatoriamente de un año de n días, y se los ordena, el número de valores duplicados entre los espacios entre esos cumpleaños ordenados debe ser una distribución Poisson con parámetro $\lambda = m^3/(4n)$.

Vía simulación permitió que los autores encontraron valores de m y n para los cuales la distribución Poisson parece satisfactoria. Los usados en la prueba son $m = 4096$ para un año de $n = 232$ días con $\lambda = 4$.

El criterio usado para calificar si un determinado GNSA pasa una prueba es que el valor estadístico resultante v que sea superior a 0.01 y menor a 0.99.

- **El test del gorila**

La idea es producir una secuencia de letras de un cierto alfabeto, para entonces estudiar la frecuencia de esas k-letras en la secuencia.

Usar secuencias muy largas sirve para descubrir debilidades como las presentes en los generadores con períodos cortos que son incapaces de producir todas las posibles k-$t$uplas.

Una manera posible eficiente de implantar estas ideas es tratar de contar el número de letras que no aparecen en las k-uplas producidas por el gorila.

Mediante simulación se determinó que dicho valor es aproximadamente normal con media teórica (24687971) y varianza empírica 4170 . Esto cuando se toma 1 bit, en una posición fija, de los enteros de 32bits generados y se usan 226 + 25 bits; el número x de palabras de 26bits es aproximadamente normal con esos parámetros.

5.4 TESTU01

L'Ecuyer and Simard (2007)

http://simul.iro.umontreal.ca/testu01/tu01.html

TestU01

TestU01 is a software library, implemented in the ANSI C language, and offering a collection of utilities for the empirical statistical testing of uniform random number generators.

The library implements several types of random number generators in generic form, as well as many specific generators proposed in the literature or found in widely-used software. It provides general implementations of the classical statistical tests for random number generators, as well as several others proposed in the literature, and some original ones. These tests can be applied to the generators predefined in the library and to user-defined generators. Specific tests suites for either sequences of uniform random numbers in [0,1] or bit sequences are also available. Basic tools for plotting vectors of points produced by generators are provided as well.

Additional software permits one to perform systematic studies of the interaction between a specific test and the structure of the point sets produced by a given family of random number generators. That is, for a given kind of test and a given class of random number generators, to determine how large should be the sample size of the test, as a function of the generator's period length, before the generator starts to fail the test systematically.

The documentation with a description of the functions in TestU01 is available in the user's guide below.

TestU01-1.2.3

This version was created on 18 August 2009.

- Licence and copyright.
- Source files (zip archive) The installation uses **configure**.
- Binaries for Cygwin under MS Windows
- Binaries for MinGW under MS Windows
- Installation
- User's guide (pdf)
- Paper (pdf) describing TestU01 with results from our test suites applied on several popular generators: P. L'Ecuyer and R. Simard, *TestU01: A C Library for Empirical Testing of Random Number Generators* ACM Transactions on Mathematical Software, Vol. 33, article 22, 2007.
  ERRATUM: *The period of generator* Brent-xor4096s *in Table I should be 2^4128 and not 2^131072.*

ÍNDICE

6. Información adicional

Dos sitios excelentes y actualizados con abundante información sobre números aleatorios son las páginas de Hellekalek http://random.mat.sbg.ac.at y de L'Ecuyer http://www.iro.umontreal.ca/~lecuyer.

Hay varios sitios que proporcionan generadores verdaderos de números aleatorios.

- http://www.random.org, que usa ruido atmosférico;
- http://random.hd.org/index.html (Java EntropyPool), que emplea ruido proveniente de entradas en distintos sitios web y otros dispositivos físicos.

Las revistas *Communications of ACM* y *Applied Statistics* publican a menudo nuevos algoritmos de generación de números aleatorios. En particular, los de esta última, pueden obtenerse de Statlib en http://lib.stat.cmu.edu/.

La librería científica GNU tiene una sección entera dedicada a números aleatorios http://www.gnu.org/software/gsl/manual/html$_$node/Random-Number-Generation.html.

Referencias

Bays, C. y Durham, S. (1976), Improving a Poor Random Number Generator, ACM Trans. Math. Soft., **2**, 59-64. (disponible en Aula Virtual)

Bowman, K. y Robinson, M. (1987), Studies of Random Number Generators for Parallel Processing, en Heath (ed), Proc. 2nd. Conf. on Hypercube Multiprocessors, SIAM, 445-453.

Couture, R. y L'Ecuyer, P. (1997) Distribution Properties of Multiply-With-Carry Random Number Generators, Mathematics of Computation **66**, 591-607. (disponible en Aula Virtual)

Deng, L.-Y. y Lin, D.K.J. (2000), Random Numbers Generation for the new Century, The American Statistician **54**, 2, 145-150.

Eddy, W. (1990), Random Number Generators for Parallel Processors, Jour. Comp. Appl. Mathematics **31**, 63-71.

Eichenauer-Herrman. J. (1995), Inversive Congruential Pseudorandom Number Generators, Int. Stat. Rev. **63**, 167-176.

Hacking, I. (1965), The Logic of Statistical Infernce, Cambridge UP.

Hellekalek, P. y Wegenkittl, S. (2003), Empirical Evidence Concerning AES, ACM Trans. On Modelling and Computer Simulation, **13**, 322-333. (disponible en Aula Virtual) .

Knuth, D. (1981), The Art of Computer Programming, Vol 2: Seminumerical Algorithms, Add. Wesley.

Kolmogorov, A. y Uspenskii, V. (1987), Algorithms and Randomness, Theor. Applic. Prob. **32**, 389-412.

L'Ecuyer, P. (1988), Efficient and Portable Combined Random Number Generators, Comm. ACM **31**, 742-749.

L'Ecuyer, P. (2006), Uniform Random Generation, en Handbook of Operations Research and Management Science. Simulation, S. Henderson and B. Nelson (eds.), 55-78. (disponible en Aula Virtual)

L'Ecuyer, P. y Simard, R. (2007), TestU01: A C Library for Empirical Testing of Random Number Generators, ACM Trans. Mathematical Software **33** (4), 22-40. (disponible en Aula Virtual)

Lehmer, D. (1951), Mathematical Methods in Large-Scale Computing Units, Proc. 2nd Symp. Large Scale Digital Calculation, Harvard U.P.

Lewis, T. y Payne, P. (1994), Uniform Random Number Generators, J. Comp. Mach. **12**, 83-89.

MacLaren, M. y Marsaglia, G. (1965), Uniform Random Number Generators, Jour. Assoc. Comp. Mach. **12**, 83- 89.

Marsaglia, G. (1968), Random Numbers Fall Mainly in the Planes, Proc. Nat. Acad. Sci. **61**, 25-28.

Marsaglia, G. (1977), The Squeeze Method for Generating Gamma Variates, Comp. Math. Appl**.** **3**, 321-325.

Marsaglia, G. (1985), A Current View of Random Number Generators, en Billard (ed) Computer Science and Statistics, Proc. 16th Symp. on the Interface, North Holland, 3-10.

Marsaglia, G. (2002), Good 64-bit RNG’s. Posted to the electronic billboard sci.crypt.random-numbers.

Marsaglia, G. (2003), Xorshift RNGs, J. Stat. Soft **8**, 1-6. (disponible en Aula Virtual)

Marsaglia, G., Ananthanarayanan, K. and Paul, N. (1973). How to Use the McGill Random Number Package SUPER-DUPER. Technical report, School of Computer Science, McGill University, Montreal, Canada.

Marsaglia, G. y Zaman, A. (1991), A New Class of Random Number Generators, The Annals of Applied Probabilities **1**, 462-480. (disponible en Aula Virtual)

Marsaglia, G. y Tsang W.W. (2002), Some Difficult-to-Pass Tests of Randomness, J. Stat. Soft **7**, 1-8. (disponible en Aula Virtual)

Matsumoto, M. y Nishimura, T. (1998), Mersenne Twister: a 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator". ACM Trans. on Modeling and Computer Simulation **8** (1), 3-30. (disponible en Aula Virtual)

Niederreiter, H. (1992), Random Number Generation and Quasi-Monte Carlo Methods, SIAM.

Park, S. y Miller, K. (1988), Random Number Generators: Good Ones Are Hard to Find, Comm. ACM **31**, 1192-1201. (disponible en Aula Virtual)

Press, W. H., Teukolvsky, S. A., Vetterling, W. T. y Flannery, B. P. (1992), Numerical Recipes in FORTRAN: the Art of Scientific Computing, Cambridge University Press.

Press, W. H., Teukolvsky, S. A., Vetterling, W. T. y Flannery, B. P. (2007), Numerical Recipes: The Art of Scientific Computing (C++), Third edition, Cambridge University Press. (disponible en Aula Virtual)

Ríos, S. (1980), Métodos Estadísticos, McGraw Hill.

Ríos Insua, D., Ríos Insua, S., Martín, J. y Jiménez, A. (2008), Simulación: Métodos y Aplicaciones, RA-MA, Madrid.

Ripley, B. (1987), Stochastic Simulation, Wiley.

Rukhin, A., Soto,J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., San Vo. (2001), NIST 800-22 “A Statistical Test Suite For random and Pseudorandom Number Generators for Cryptographic Applications”.

Schruben, L. y Margolin, B. (1978), Pseudo-random Number Assignment in Statistically Designed Simulation and Distribution Sampling Experiments, Jour. Amer. Stat. Assoc. **73**, 504-525.

Tausworthe, R. (1965), Random Numbers Generated by Linear Recurrence Modulo Two, Math. Comp. **19**, 201-209. (disponible en Aula Virtual)

Thompson, J.R. (1999), Simulation: A Modeler's Approach, Wiley.

Wichmann, B.A. y Hill, I.D. (1982), Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator, Applied Statistics **31** (2) 188-190. (disponible en Aula Virtual)