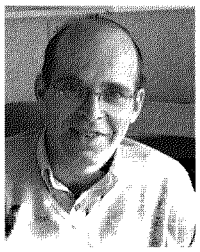
Revista Española de Física **16** (2), 2002

ANGEL SÁNCHEZ

Se pueden enviar contribuciones a esta sección siguiendo el procedimiento habitual de REF o bien directamente a anxo@math.uc3m.es en formato LaTeX, PostScript o PDF.

Continuando con la pequeña serie que anunciábamos en el número anterior, en éste presentamos dos artículos sobre generación de números aleatorios. Por un lado, Raúl Toral nos da un (inevitablemente) breve resumen de lo que se sabe sobre el tema; por otro, Santos Bravo Yuste y Héctor Sánchez Pajares nos ponen un ejemplo muy sorprendente de cómo los generadores de números aleatorios pueden jugárnosla de manera muy sutil. Creo que no necesito recalcar la utilidad que ambos trabajos, de carácter muy práctico, pueden tener para los lectores de REF que se "ensucian las manos" con este tipo de simulaciones o que quieran empezar a hacerlo. Para el próximo número concluiremos la serie hablando sobre los efectos constructivos, ordenadores, del ruido.

Archivo de la sección:
http://gisc.uc3m.es/fisica_y_computacion



Los números aleatorios en Sevilla son una maravilla

RAÚL TORAL

Un verdadero número aleatorio es el producido por un proceso aleatorio natural: una desintegración radiactiva, por ejemplo. Así, el número de electrones emitidos por unidad de tiempo por una sustancia radiactiva beta sigue una distribución de Poisson, y el tiempo entre emisiones consecutivas de dos electrones sigue una distribución exponencial. Sería fácil utilizar estas dos distribuciones para generar cualquier otra, continua o discreta, necesaria en la simulación de ecuaciones estocásticas o cualquier otro proceso que los requiriera. Sin embargo, es obvio que esto es algo claramente ineficiente y que necesitamos de algún algoritmo fácilmente programable que nos permita obtener los números aleatorios necesarios de una manera rápida y, a la vez, precisa. Aquí nos encontramos con una contradicción que a, la larga, resulta ser una ventaja: no es posible escribir un algoritmo de ordenador que nos dé números realmente aleatorios. Por definición, un número aleatorio es impredecible, mientras que el resultado de un programa de ordenador es reproducible: cada vez que corramos el programa obtendremos los mismos resultados. Es claro que, al menos en la fase de desarrollo de un programa, es conveniente tener un algoritmo que dé siempre los mismos números aleatorios si no queremos volvernos locos buscando errores que dan resultados no deseados. En la fase de producción del programa, uno podría pensar que es conveniente introducir un elemento de aleatoriedad en el programa y a veces se utiliza un proceso físico subyacente, siendo el ejemplo más utilizado el de una variable que dependa del tiempo que lleve la máquina en funcionamiento. La introducción de este grado adicional de aleatoriedad no es en general aconsejable ya que no permite controlar la reproducibilidad de los resultados (y uno podría acabar publicando los resultados en el *Journal of Irreproducible Results*, véase la página web http://www.jir.com).

Si disponemos de un generador de números aleatorios u distribuidos uniformemenente en el intervalo (0,1), podemos generar cualquier otra distribución de una variable $f(x)$ [1]. No hay más que considerar la función cumulativa $F(x)=\int_{-\infty}^{x} dx' f(x')$ e invertir $x=F^{-1}(u)$. Conseguimos así que los números x estén distribuidos según la función densidad $f(x)$. En aquellas ocasiones en las que la inversión de la función $F(x)$ es difícil de hacer de forma analítica exacta, se puede recurrir a métodos numéricos suficientemente precisos. Para distribuciones concretas es posible desarrollar métodos específicos. Quizás el más famoso sea el de Box-Muller-Wiener para generar números aleatorios gaussianos de media cero y varianza uno. A partir de un cambio a coordenadas polares es posible demostrar que g_1, g_2 definidos por

$$g_1 = \sqrt{-2\ln u_1}\,\cos(2\pi u_2)$$
$$g_2 = \sqrt{-2\ln u_1}\,\sin(2\pi u_2) \qquad (1)$$

son números gaussianos independientes (de media cero y varianza uno) si u_1, u_2 son números independientes distribuidos uniformemente en el intervalo (0,1). Se ha hecho mucho uso de este método de Box-Muller-Wiener, aunque es mucho más eficiente utilizar un método de inversión numérica de la función cumulativa $F(x)$ que involucra la función error [2]. Un número aleatorio gaussiano z de media μ y varianza σ^2 se obtiene mediante la transformación $z=\sigma x+\mu$.

Aunque es usual utilizar la caja negra que representa el generador de números aleatorios que viene con el compilador, en la confianza de que alguien lo habrá probado y será bueno, la verdad es que conviene desconfiar (en el otro artículo de esta sección, de Santos Bravo Yuste y Héctor Sánchez-Pajares [3], tendremos ocasión de comprobar como un generador que ha sido dado por bueno en muchas ocasiones, puede darnos resultados tremendamente malos en otras).

En general, en aquellas aplicaciones en las que uno quiera ser cuidadoso con los resultados y sus barras de error, es necesario tener un completo control del generador e, incluso, repetir las simulaciones con dos generadores distintos para ver que los resultados coinciden.

Para generar números uniformes en el intervalo (0,1) se utilizan, casi sin excepción, números de la forma $u_i=m_i/M$ donde M es un número entero suficientemente grande y $m_i \in (0,M-1)$, $i=0,1,2,...$, aunque algo mejor es usar $u_i=(m_i+0.5)/M$ que evita que aparezca el valor $u_i=0$, lo que es conveniente, por ejemplo, a la hora de usar el algoritmo de Box-Muller-Wiener, donde hay que calcular ln u. Los valores de m_i están escogidos de manera que los números resultantes u_i satisfagan los tests de aleatoriedad necesarios. Dicho de otra manera: si a un matemático le damos una secuencia de números u_0, u_1, u_2,... obtenidos de esta manera, ha de ser incapaz de discernir si los números son realmente aleatorios o si provienen de algún algoritmo de ordenador. Para satisfacer el requisito $m_i \in (0, M-1)$ se suele generar mediante un algoritmo del tipo $m_{i+1}=F(m_i)$ mod M con una función $F(m)$ apropiada, empezando de un número *semilla* m_0 cualquiera. Hay un famoso algoritmo desarrollado por von Neumann que usa $M=10^4$ y la función $F(m)=[m^2/100]$ ($[x]$ designa la parte entera de x). Este algoritmo es muy malo aunque no sea más que por el hecho de que al usar $M=10^4$ cualquier secuencia de más de 10^4 números consiste necesariamente de secciones de longitud M repetidas. M se denomina el periodo del número aleatorio. Aparte de que el periodo máximo es tan pequeño, no es raro encontrarse con periodos menores o incluso con puntos fijos $m_{i+1}=m_i$ que hacen que el algoritmo de von Neumann quede reducido a una curiosidad histórica.

Mejores resultados se obtienen usando una función lineal $F(m)=a\,m+c$ y escogiendo adecuadamente los números M, a y c. En la literatura encontramos valores de a, c y M que alguien asegura que ha probado y le van bien (!), todo ello sin demasiada justificación. Como mínimo, se procura que el periodo del generador sea igual al valor máximo M, para lo que es suficiente que $M=2^b$ sea una potencia de dos, que c sea impar y que $a=1$ mod 4. El que M sea una potencia de dos tiene la ventaja de que la operación módulo es muy rápida de hacer ya que basta con truncar los primeros bits de la representación binaria de m para quedarse con los b últimos bits. Valores dados por "buenos" (con todas las reservas expresadas con anterioridad) son, $M=2^{32}$, $c=1$, $a=1812433253$. Entre los fabricantes de compiladores parece haberse generalizado el uso de $M=2^{32}$, $c=1$, $a=69069$ cuya principal virtud parece ser la de ser fácil de recordar.

Vamos a dar una implementación de la función *rand(iseed)* que genera un número pseudoaleatorio uniformemente distribuido entre 0 y 1 usando el algoritmo precedente y utilizando el que estamos en un ordenador de 32 bits.

```
function rand(iseed)
parameter (rm=2.0**32,ia=1812433253, ic=1)
iseed=iseed*ia+ic
rand=rm*(iseed+0.5)
if (rand.lt.0.0) rand=1.0+rand
return
end
```

Para que este programa funcione es imprescindible el que el ordenador no detecte el overflow que puede existir en la multiplicación entera. Para ello puede ser necesario darle la instrucción adecuada al compilador. Cuando ello no sea posible, es necesario reprogramar el algoritmo usando aritmética real de doble precisón, que tiene la desventaja de ser más lento.

En los últimos años se ha venido generalizando el uso de otra familia de número aleatorios que generan bits aleatorios $z_i = 0,1$. Estos algoritmos, llamados FSR de ``feedback shift register", están basados en la relación de recurrencia $z_i = c_1\, z_{i-1} + c_2\, z_{i-2} + ... + c_n\, z_{i-n}$ mod 2 donde $c_i = 0,1$ es una conjunto de n constantes binarias. Es claro que esta secuencia de z_i ha de repetirse necesariamente después de 2^n valores, de manera que el periodo máximo es 2^n. Para alcanzar este periodo máximo se debe verificar que el polinomio $f(z)=1+c_1\, z+c_1\, z^2 + \cdots + c_n\, z^n$ no se pueda descomponer como $f(z)=f_1(z)f_1(z)$ (todas las operaciones módulo 2). Resultados suficientemente buenos se obtienen tomando la sencilla relación de recurrencia $z_i = z_{i-p} + z_{i-q}$ mod (2) que tiene la ventaja de que se puede escribir usando la operación lógica "o exclusivo": $z_i = z_{i-p} \otimes z_{i-q}$ que es extremadamente eficiente de implementar en un ordenador. Una vez hemos conseguido una secuencia de M bits aleatorios, estos se juntan para formar el número m_i a partir del cual se obtiene el número uniforme u_i deseado. La pareja de valores $p=250$, $q=103$ estuvo de moda durante un tiempo y el generador resultante se bautizó como R250. Sin embargo, se han encontrado algunos problemas con esta pareja de números y se recomienda que se utilice $p=1279$, $q=418$. El periodo de este generador es $2^{1279} \approx 10^{385}$, que es inalcanzable en cualquier simulación actual (y futura).

Es conveniente dar un último consejo: los números aleatorios generados mediante un algoritmo determinista no son nunca perfectos. La obviedad de que algunos generadores son mejores que otros tiene una versión pesimista compartida por muchos: todos los generadores son malos, pero algunos son peores que otros. Siempre existen correlaciones entre los números generados por los algoritmos explicados anteriormente (o por cualquier otro). Estas correlaciones suelen ser muy sutiles y aparecer únicamente en determinados casos o en problemas con una dimensión espacial determinada o para un tamaño especial del sistema. Por ejemplo, el teorema de Marsaglia [4] afirma que existe una dimension d tal que los números aleatorios $(u_i,\cdots,u_{i+d})$ generados por los generadores congruenciales o FSR, en vez de llenar el espacio $[0,1]^d$, caen en un hiperplano de dimensión inferior a d.

Para resumir, permitáseme recomendar que es esencial tener un control completo del generador usado en nuestras simulaciones en vez de confiar en la caja negra que viene con el compilador. En caso de resultados sospechosos, hay que comprobar que otro generador da los mismos resultados.

Bibliografía

[1] El caso de varias variables dependientes es mucho más difícil y, en los casos más complicados, conviene recurrir a los métodos de Monte Carlo, fuera del alcance de este corto artículo.

[2] Un algoritmo explícito que usa el método de inversión numérica para general números aleatorios gaussianos se puede encontrar en R. Toral y A. Chakrabarti, *Comp. Phys. Comm.* **74**, 327 (1993).

[3] S. Bravo y H. Sánchez-Pajares, *Revista Española de Física*, artículo siguiente.

[4] Este teorema apareció en el artículo *Random numbers fall mainly on the planes*, G. Marsaglia, *Proc. Nat. Acad. Sci.* **61**, 25 (1968).

Raúl Toral