9
2ej2

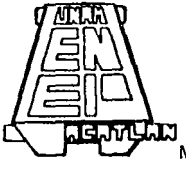
UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"A C A T L A N"

ANALISIS DE ALGUNOS GENERADORES DE NUMEROS ALEATORIOS

T E S I S

QUE PARA OBTENER EL TITULO DE:
LICENCIADA EN MATEMATICAS APLICADAS Y COMPUTACION
P R E S E N T A :
FELISA CORTES RUIZ



ASESOR DE TESIS:
M.I. NORMA ELENA URIBE MEMIJE

MEXICO, D. F. NOVIEMBRE 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

UNIVERSIDAD NACIONAL AUTÓNOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "ACATLAN"

DIVISION DE MATEMATICAS E INGENIERIA
PROGRAMA DE ACTUARIA Y M.A.C.

SRITA. FELISA CORTES RUIZ
Alumna de la carrera de M.A.C.
P r e s e n t e .

Por acuerdo a su solicitud presentada con fecha lo. julio de 1994, me complace notificarle que esta Jefatura tuvo a bien asignarle el siguiente tema de Tesis: "ANALISIS DE ALGUNOS GENERADORES DE NUMEROS ALEATORIOS" el cual se desarrollará como sigue:

INTRODUCCION
CAP. I Algoritmos y números aleatorios
CAP. II Métodos para generar números aleatorios
CAP. III Pruebas estadísticas para los números aleatorios
CAP. IV Atributos de los algoritmos generadores de números aleatorios seleccionados para el estudio
CAP. V Evaluación de los algoritmos
CONCLUSIONES
ANEXOS
BIBLIOGRAFIA

Asimismo, fué designado como Asesor de Tesis la M.I. NORMA ELENA URIBE MEMIJE.

Ruego a usted tomar nota que en cumplimiento de lo especificaco en la Ley de Profesiones, deberá presentar servicio social durante un tiempo mínimo de - - seis meses como requisito básico para sustentar examen profesional así como de la disposición de la Coordinación de la Administración Escolar en el sentido de que se imprima en lugar visible de los ejemplares de la Tesis el título del trabajo realizado. Esta comunicación deberá imprimirse en el interior de la Tesis.

A T E N T A M E N T E
"POR MI RAZA HABLARA EL ESPIRITU"
Acatlán, Edo. Méx. noviembre 10 de 1994.

ACT. LAURA MA. RIVERA BECERRA
Jefe del Programa de Actuaría
y M.A.C.

E.N.E.P. ACATLAN
JEFATURA DEL PROGRAMA DE ACTUARIA Y APLICACIONES

cg'

GRACIAS A DIOS

A MIS PADRES
Por la gran confianza
que siempre han depositado
en mí.

A MIS HERMANOS
Por todo el apoyo brindado

Y a todas aquellas personas que de alguna manera
han contribuido al logro de este objetivo

I N D I C E

OBJETIVO:

Proporcionar alternativas que permitan lograr el funcionamiento óptimo de los siete algoritmos analizados.

HIPOTESIS:

Si los siete algoritmos tienen aleatoriedad y uniformidad entonces pueden aceptarse como generadores de números aleatorios y ser integrados en un paquete de computación para su explotación.

INTRODUCCION

La importancia de los números aleatorios en el área de la Simulación entendida como el diseño de modelos que representan fenómenos reales; ha originado que importantes científicos dediquen sus estudios al área de la generación de números aleatorios.

El hablar de números aleatorios, invita a pensar en los juegos de azar, por ejemplo el lanzamiento de dados. Al lanzar un dado se desconoce el número que caerá, es decir, no podemos asegurar que en nuestra tirada obtendremos un 3 o que obtendremos un 5, por lo que decimos que nuestro resultado es un número aleatorio. Este es un método manual para generar números aleatorios, aunque resulta poco práctico cuando deseamos obtener una cantidad grande de números. Ante esta situación se utilizan relaciones de recurrencia con las que se ha probado que se generan secuencias de números con distribución uniforme y estadísticamente independientes.

En este trabajo se exponen siete algoritmos que utilizan reglas de recurrencia para la generación de números aleatorios entre las cuales la más conocida es la de Lehmer. Es sabido que existe una amplia gama de investigaciones en esta materia; los algoritmos que se incluyen aquí corresponden a los investigadores: Brian Wichmann y David Hill, Pierre l'Ecuyer, G. Marsaglia, A. Zaman, W.-W. Tsag y Howard L. Kaplan. Se analizan estos algoritmos en base a la información que cada autor nos proporciona en su artículo respectivo, y a través de los resultados obtenidos al analizar las secuencias generadas directamente por los algoritmos una vez programados.

El estudio se realiza considerando los elementos que caracterizan a un buen generador de números aleatorios, a partir de esta información se determina si un algoritmo presenta alguna deficiencia y se propone alguna alternativa que lleve a la corrección de esta.

Estos siete algoritmos se han integrado en un paquete de computación, programado en PASCAL, como un primer módulo; como segundo módulo se han agregado cinco procesos para generar números aleatorios con otras distribuciones de probabilidad como la Binomial, Poisson, Normal y Weibull, incluyendo la uniforme para un intervalo (a,b); a fin de que puedan ser utilizados para obtener secuencias de números aleatorios.

CAPITULO 1

ALGORITMOS Y NUMEROS ALEATORIOS

1.1 CONCEPTO DE ALGORITMO

La palabra algoritmo se deriva de la traducción al latín de la palabra árabe Alkhowârizmî, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX (Joyanes, 1988).

Un algoritmo es una serie de pasos ordenados que nos ayudan a resolver un problema [1].

Siempre que intentemos resolver un problema, teniendo o no como herramienta la computadora, definimos la solución por medio de una secuencia de acciones que será preciso efectuar para obtener un resultado satisfactorio, en estos momentos estamos construyendo, de manera inadvertida, un algoritmo.

1.1.1 Características de un algoritmo

Las características necesarias para el diseño de un algoritmo son las siguientes:

A). Entradas y salidas

Deben estar claramente definidas las salidas (resultados que deseamos obtener) y las entradas (información con la que contamos para resolver un problema) que nos han de conducir a dichas salidas.

[1] Esquivel,Deschams, et al., "Apuntes de programación y computadoras", UNAM, 1982.

B). Precisión

Los pasos del algoritmo no deben contener ambigüedades, es decir, cada uno de ellos debe estar claramente definido,y se debe indicar el orden en el que deben realizarse. También es de gran importancia indicar acciones opcionales para cualquier situación que se presente, al momento de estar resolviendo el problema.

C). Definición y efectividad

Al momento de seguir un algoritmo por segunda vez con los mismos datos de entrada, obtendremos los mismos resultados, es decir, nuestro algoritmo está definido; además siempre nos proporciona al menos una solución del problema, en esto consiste su efectividad.

D). Finitud

Por último, se deben especificar y definir con precisión en que momento concluirán los cálculos del algoritmo. Es decir, la solución debe ser obtenida en un número finito de pasos.

Si nuestro algoritmo satisface estas características estamos logrando una descripción clara y fácil de seguir; esto es muy útil para su desarrollo completo, es decir; su transformación en programa y su documentación será más sencilla.

1.1.2 Clasificación de los algoritmos

Una forma de clasificar los algoritmos es de acuerdo al tipo de datos que maneja para resolver el problema:[2]

A). Numéricos

La solución del problema implica una serie de decisiones y cálculos que requieren de información numérica. Se dirigen a la solución de problemas científicos o aquellos que de alguna manera involucran cálculos matemáticos. Un ejemplo sencillo es, determinar la lista de números primos menores a un número dado. Los pasos a seguir son los siguientes:

i. Verificar que el número proporcionado sea un valor entero.

ii. Si es mayor o igual a 2, 2 es el primer elemento de la lista; si es igual, también es el único y se ha terminado el proceso.

iii. Si es mayor o igual a tres, 3 es el segundo elemento de la lista; si es igual, también es el último y se ha terminado el proceso; si es mayor, el número 3 lo asignamos a una variable llamada NUMERO.

iv. Sumar a NUMERO dos unidades (al hacer esto descartamos todos los números pares).

v. Si el número proporcionado es mayor o igual a NUMERO, continuar con el paso 6; si no lo es, el proceso ha terminado.

vi. Calcular la raíz cuadrada de NUMERO; si el residuo es cero, NUMERO no es primo y pasamos al paso 4; si no, continuamos con el paso 7.

vii. Verificar que la división de NUMERO entre cada uno de los elementos de la lista, menores a la parte entera de la raíz cuadrada de NUMERO, dé como residuo un valor diferente de cero. Si

[2] Esquivel, Deschams, et al. op. cit. .

para alguno no se cumple, NUMERO no es primo y pasamos al paso 4. De lo contrario NUMERO es el siguiente elemento de la lista de números primos y pasamos al paso 4.

B). No numéricos

En nuestra vida diaria desarrollamos actividades que necesariamente requieren de una secuencia de pasos que se realizan en forma ordenada, en otras palabras, necesitan de un algoritmo. Por ejemplo, al momento de trasladarnos de nuestro hogar hacia el lugar donde laboramos. Los pasos a seguir serían:

i. Asegurarnos de contar con el dinero necesario para el pasaje o abstenernos de viajar.

ii. Llegar a la parada del autobús y esperar a que este aparezca.

iii. Si es el autobús de la ruta adecuada abordarlo; si no, seguir esperando.

iv. Al abordar el autobús pagar nuestro pasaje y esperar nuestro cambio, en caso de ser necesario.

v. Si hay asientos disponibles viajar sentados; si no, viajar de pie.

vi. Cuando el autobús se aproxime a nuestro destino acercarnos a la puerta de bajada, sino es así, continuar.

vii. Descender del autobús y caminar hacia nuestro lugar de trabajo.

Este es un ejemplo claro y sencillo de un algoritmo no numérico, en el que no se utilizan valores numéricos; sólo se emplean cadenas de caracteres (símbolos) alfabéticos.

1.1.3 Pasos para desarrollar un algoritmo

Un algoritmo nos proporciona la solución de un problema mediante una serie de actividades. A su vez, para construir un algoritmo es necesario tomar en cuenta estos pasos fundamentales:

A. Análisis del problema.

Este primer paso es de gran importancia para lograr resolver nuestro problema en forma satisfactoria.

Antes de poder comprender un problema, debemos ser capaces de dar una formulación precisa. Esta condición no es, en sí, suficiente para entender el problema, pero es absolutamente necesario.

Desarrollando una formulación precisa del problema es importante plantearse ciertas preguntas. Algunas buenas preguntas sobre lo relacionado con la formulación del problema son:

¿Entiendo el vocabulario usado en la formulación?

¿Qué información se da?

¿Qué es lo que se desea encontrar?

¿Cómo conozco una solución?

¿Qué información falta, se usará algo de esta información?

¿Alguna información que se da es inútil?

¿Qué supuestos se han establecido?

Pueden plantearse otras preguntas dependiendo del problema en particular. Con frecuencia es necesario plantearse de nuevo algunas de estas preguntas, una vez que han sido contestadas parcial o totalmente.

Esto es una formulación básica buena del problema que nos permite saber lo que tenemos y lo que deseamos encontrar.

B. Diseño del algoritmo.

Una vez que se ha definido qué se va a hacer, estamos en condiciones de establecer cómo vamos a hacerlo; ahora podemos determinar los pasos que hemos de seguir para lograr nuestro

objetivo: la solución del problema.

Durante los años de la computación los científicos han identificado un número de técnicas generales que con frecuencia producen algoritmos efectivos en la solución de diversas clases de problemas. Entre las técnicas más importantes se encuentran: divide y vencerás, programación dinámica, técnicas "greedy" , "backtracking" y búsqueda local (Aho, Hopcroft, Ullman, 1983). Tratando de planear un algoritmo para resolver un problema dado, resulta útil plantearse preguntas tales como ¿Qué tipo de soluciones produce divide y vencerás, programación dinámica, una aproximación "greedy", o alguna otra técnica estandar?.

Debe enfatizarse, sin embargo, que existen problemas, tales como los problemas "NP-completo" (problemas en los cuales todas las soluciones conocidas son esencialmente del tipo "trata todas las posibilidades"), para los cuales estas o cualquier otra técnica conocida no proporcionará soluciones eficientes. Cuando se encuentra un problema tal, es útil determinar si las entradas del problema tienen características especiales que podrían explotarse tratando de planear una solución, o si podría usarse una solución aproximada más fácil de encontrar en lugar de la solución exacta difícil de calcular.

C. Codificar el algoritmo.

Consiste en representar el algoritmo en un lenguaje de programación, es decir, expresarlo a través de un conjunto de símbolos y palabras especiales aceptados por la computadora (programa).

D. Verificación del programa.

Probar el programa para varios casos cuya solución se puede comprobar fácilmente, analizar los resultados y en caso de ser incorrectos efectuar las modificaciones y correcciones pertinentes.

E. Documentación.

El programa debe poder ser utilizado por cualquier persona, debido a esto, es muy necesario escribir un instructivo detallado sobre el funcionamiento del programa, alcances y limitaciones.

Al hablar aquí de un programa nos referimos a la descripción que se está haciendo del algoritmo en algún lenguaje, es decir, un programa es la expresión que se le dá al algoritmo para que la computadora pueda realizar las tareas definidas en él (Joyanes,1988).

F. Implementación del programa.

Una vez que se han desarrollado los pasos anteriores estamos preparados para alimentar nuestro programa con la información del problema para el que se ha desarrollado nuestro algoritmo.

El lenguaje de programación es un conjunto de reglas, símbolos y palabras especiales (un lenguaje), que permite construir un programa de computadora que pueda ser entendido por ella. Existe gran diversidad de lenguajes de programación, con sus ventajas y desventajas; orientados hacia fines específicos, para resolver cierto tipo de problemas, de educación, científicos, de investigación, etc. (Joyanes, 1988).

Los lenguajes de programación se clasifican en tres categorías: máquina, bajo nivel y alto nivel.

Lenguaje máquina.

Los lenguajes máquina son aquellos cuyas instrucciones comprende directamente la computadora, es decir, no necesita traducción para que la máquina pueda ejecutarlo. Depende de la estructura física ó electrónica de una computadora; utilizan un lenguaje binario (0 y 1), que es el único que entiende la máquina. Estos suelen ser complicados, largos y difíciles de entender por el programador.

Lenguajes de bajo nivel (ensamblador).

Los lenguajes de bajo nivel tratan de simplificar el proceso de programación en lenguaje máquina, generalmente dependen del tipo de máquina en que se esté programando; es decir, depende de un conjunto de instrucciones específicas propios de la máquina.

En este lenguaje, las instrucciones se escriben en códigos alfabéticos conocidos como nemotécticos (abreviaturas de palabras inglesas o españolas). Por ejemplo:

ADD	*sumar*	MPY	*multiplicar*
SUB	*restar*	DIV	*dividir*

Para el programador, estas instrucciones resultan ser más fáciles de comprender y aplicar que las secuencias de 0 y 1.

Lenguajes de alto nivel.

Los lenguajes de alto nivel permiten el desarrollo de programas sin conocer el funcionamiento interno de la máquina, es decir, no dependen del tipo de máquina. En estos lenguajes (por ejemplo: el ADA, BASIC, Modula-2, Pascal, etc.), sus instrucciones o sentencias son escritos con palabras similares a los lenguajes humanos, generalmente el inglés, facilitando la escritura y comprensión por el programador.

Este tipo de lenguajes, al no depender de la máquina son

generalmente transportables, esto es, un programa escrito en alguno de estos lenguajes se puede escribir con poca o ninguna modificación en diferentes tipos de máquina.

Para el programador resulta fácil y sencillo escribir un programa en lenguaje de alto nivel, pero para que la máquina pueda ejecutarlo es necesario traducirlo a lenguaje máquina; los programas que realizan estas tareas son los compiladores e intérpretes que generalmente son llamados traductores.

Un programa escrito en lenguaje de alto nivel se llama programa fuente y el que tiene la expresión en lenguaje máquina se llama programa objeto.

1.2 CONCEPTO DE NUMERO ALEATORIO

En la actualidad existe gran variedad de temas que de alguna manera involucran la teoría de probabilidad. Por ejemplo, en genética se realizan predicciones sobre la frecuencia relativa con la que aparecen ciertas características en un grupo de individuos (es difícil que se dé una misma proporción aún en miembros de la misma familia); no es posible predecir con exactitud la densidad de tráfico automovilístico en un cruce de caminos a determinada hora del día; los ingenieros industriales aplican la probabilidad al tratar de mantener un nivel de calidad determinado en los productos manufacturados; etc.. Estos fenómenos poseen la característica de que al ser observados bajo condiciones idénticas, en cada observación su resultado es diferente.

Un ejemplo típico en el que identificamos claramente la característica de aleatoriedad es el siguiente: Supongamos que tenemos una urna con bolas blancas y azules, el experimento consiste en extraer una bola de la urna y observar su color. Al realizar el experimento en estas condiciones obtendremos algunas veces bolas blancas y otras, bolas azules; significa que nosotros no podemos saber con exactitud de que color será la bola que extraigamos en un momento dado. Un fenómeno con esta característica se llama fenómeno aleatorio.

Si repetimos el experimento N veces bajo las mismas condiciones y de estos resultados N_b veces se extrae una bola blanca (resultado al que le llamaremos atributo b), a la razón de N_b/N le llamamos **frecuencia relativa**[3] .

El conjunto formado por todos los resultados posibles del experimento aleatorio se llama **espacio muestral**. De este espacio muestral pueden interesarnos los resultados que tengan alguna característica en común. Hecho que nos permite dar la siguiente definición: Un **evento** del espacio muestral es un grupo de resultados contenidos en éste, cuyos elementos poseen una característica en común.

Cuando en un experimento existe un evento del que tenemos la certidumbre de que ocurrirá, decimos que este es un **evento seguro**.

Por otra parte el evento que no contiene algún resultado del espacio muestral lo llamamos **evento vacío**.

Nos será de gran ayuda tener presentes algunas nociones de teoría de conjuntos por lo que a continuación damos algunas definiciones importantes.

Sean E_1 y E_2 dos eventos cualesquiera que se encuentran en el espacio muestral denotado por S.

El evento formado por todos los posibles resultados en E_1 o E_2 o en ambos, recibe el nombre de **unión** de E_1 y E_2, denotado como $E_1 \cup E_2$.

El conjunto que se forma de los resultados comunes de E_1 y E_2, se llama **intersección** de E_1 y E_2 y se denota por $E_1 \cap E_2$.

Dos eventos E_1 y E_2 son **mutuamente excluyentes o disjuntos** si no contienen resultados en común; es decir, $E_1 \cap E_2 = \phi$ evento vacío.

Una vez que se han expuesto los conceptos básicos de teoría de probabilidad estamos preparados para dar la siguiente:

[3] Parzen, Emanuel. "Teoría moderna de probabilidades y sus aplicaciones". México, Limusa-Wiley, 1971.

Definición: Sean S cualquier espacio muestral y E cualquier evento de éste. Llamamos **función de probabilidad** sobre el espacio muestral S a **P(E)** si satisface (Canavos, 1986):

a) $P(E) \geq 0$.

b) $P(s) = 1$.

c) Si para los eventos $E_1, E_2, E_3, \ldots$

$$E_i \cap E_j = \phi \quad \text{para toda} \quad i \neq j \text{ , entonces}$$

$$P (E_1 \cup E_2 \cup E_3 \cup \ldots) = P (E_1) + P (E_2) + \ldots$$

Los conceptos de probabilidad anteriores se refieren a eventos que se encuentran en un espacio muestral. Los eventos de un espacio muestral pueden ser cualitativos ó cuantitativos. Un ejemplo de eventos cualitativos es, en el experimento de extraer una bola de una urna, obtener una bola blanca ú obtener una azul. Para estudiar el comportamiento aleatorio de un fenómeno, es de gran ayuda cuantificar los eventos del espacio muestral conforme a una medida numérica. Al establecer esta relación entre eventos y medidas cuantitativas se da origen a un nuevo concepto: variable aleatoria .

Se llama variable aleatoria a la función que asigna uno y sólo un número real a cada evento del espacio muestral[4].

Los valores de la variable aleatoria involucran la probabilidad, puesto que los eventos están asociados a este concepto.

Si el conjunto de valores tomados por la función lo forman únicamente los números enteros, la variable aleatoria se llama discreta; si esta consiste de números reales, entonces la variable aleatoria es continua.

[4] Feller Wiliam., "Introducción a la teoría de probabilidades y sus aplicaciones". México, Limusa-Wiley, 1973.

Si x es una variable aleatoria, llamamos **distribución de probabilidad** a la función $f(x)$ que establece la relación entre x y su medida de probabilidad.

Si x es una variable aleatoria continua en el rango de $-\infty$ a $+\infty$ su distribución de probabilidad $f(x)$ debe satisfacer las condiciones:

a) $f(x) \geq 0$; para $-\infty < x < \infty$.

b) $\int_{-\infty}^{\infty} f(x)dx = 1.$

Si x es una variable aleatoria discreta su distribución de probabilidad, denotada $p(x)$, debe satisfacer las condiciones:

a) $p(x) \geq 0$; para toda x.

b) $\sum_x p(x) = 1.$

En el caso continuo, $f(y)$ evalúa la probabilidad de que la variable aleatoria x tome el valor y , es decir, $f(y) = P\{x=y\}$. En el caso discreto $p(a)$, significa lo mismo: $p(a) = P\{x=a\}$.

La función de densidad acumulada, denotada $F(a)$, se define como la probabilidad de que una variable aleatoria sea igual o menor a un número real a $(x \leq a)$. Esto es,

$$F(a) = P\{x \leq a\} = \int_{-\infty}^{a} f(x)dx$$

en el caso continuo, y

$$F(a) = P\{x \leq a\} = \sum_{x=-\infty}^{a} p(x)$$

si x es variable aleatoria discreta, donde $p(y) = P\{x=y\}$ y $f(y) = P\{x=y\}$.

La función de densidad acumulada, tiene las siguientes 4 propiedades:

a) $0 \leq F(x) \leq 1$, para toda x.

b) $\lim_{x \to -\infty} F(x) = 0$.

c) $\lim_{x \to \infty} F(x) = 1$.

d) $F(x)$ es una función monótona no decreciente de x, es decir,

$F(a) \geq F(b)$ para toda variable aleatoria a y b, tal que $a \geq b$.

CAPITULO 2

METODOS PARA GENERAR NUMEROS ALEATORIOS

2.1 INTRODUCCION

La necesidad de números aleatorios para realizar investigaciones científicas u otros campos de trabajo ha estado latente desde mucho antes del advenimiento de las computadoras. Los métodos que desde entonces se empleaban para generar los números aleatorios son muy sencillos y simples, como el lanzar una moneda, dados, emplear barajas o la ruleta, o la extracción de bolas de una urna; son técnicas que denominamos como *manuales*. Con el paso del tiempo y el avance de la tecnología estas técnicas han sido desplazadas por las computacionales que si no proporcionan números exactamente aleatorios, estos presentan ajustes muy aceptables.

2.2 TECNICAS MANUALES

Los métodos utilizados para generar números aleatorios antes del surgimiento de las computadoras, tales como el lanzamiento de moneda, dados, empleo de barajas o la ruleta, o la extracción de bolas de una urna que ha sido perfectamente agitada; son métodos realmente sencillos.

Estos métodos sin lugar a duda son muy fáciles de llevar a cabo, pero muy tediosos si nuestros requerimientos de números aleatorios son muy grandes, por otra parte, al momento de repetir nuestro experimento los números generados no serán los mismos; por lo que éstos métodos, aunque se considera que generan números verdaderamente aleatorios, son muy imprácticos.

Se han llegado a elaborar tablas de biblioteca de números

aleatorios como la publicada por L. H. C. en 1927, en donde se presentaron 40,000 números aleatorios tomados al azar de reportes de censos.

A partir de ésto se ha construido una serie de dispositivos mecánicos para generar números aleatorios, en 1939 se utilizó la primer máquina, por M. G. Kendall y B. Babington-Smith para producir una tabla de 100,000 dígitos aleatorios y en 1955 la corporación *Rand* publicó una tabla de un millón de números, que fueron generados con un dispositivo especial. En la lotería British Premiun Savings Bond se usó una famosa máquina de números aleatorios llamado ERNIE para seleccionar los números ganadores.

Poco tiempo después aparecieron las computadoras y se buscaron nuevas formas para realizar esta tarea tan importante.

2.3 TECNICAS AUTOMATIZADAS

En la generación de números por computadora podemos distinguir tres métodos: la provisión externa, la generación interna con la utilización de un proceso físico y la generación interna por medio de operaciones aritméticas.

A. Provisión externa

La provisión externa consiste principalmente en el almacenaje de las tablas, ya mencionadas anteriormente, en cintas magnéticas. Este fue el primer método desarrollado por computadora. Estos números fueron utilizados como datos de entrada en la solución de un problema, pero el método requiere de gran espacio de memoria y la lectura de datos es muy lenta.

B. Generación interna por medio de procesos físicos aleatorios

Este método consiste en la utilización de un aditamiento especial de la computadora capaz de registrar los resultados de algún proceso aleatorio y reducirlo a sucesiones de dígitos.

Por ejemplo, agregar una máquina como la ERNIE a la computadora. Esta técnica tiene sus deficiencias, por una parte no es posible reproducir cálculos una segunda vez, y por otra, este tipo de máquinas con frecuencia sufren de averías difíciles de detectar.

C. Generación interna por medio de operaciones aritméticas.

Por todos los inconvenientes que presentan las técnicas anteriores surge el interés de producir números aleatorios por medio de operaciones aritméticas de una computadora.

El primer método de este tipo fue el de los *cuadrados centrales* propuesto por John von Neumann y Metropolis en 1946. Este método consiste en tomar un número aleatorio anterior y elevarlo al cuadrado, se extraen los dígitos centrales de este número y éstos forman el siguiente número aleatorio. Por ejemplo, si generamos números de 4 dígitos y el anterior fué 5698 , su cuadrado es

32467204

entonces el siguiente número aleatorio es: 4672.

Este tipo de métodos generan los números por medio de una relación de recurrencia por lo que se presenta la siguiente objeción: ¿cómo pueden ser aleatorios estos números si son generados en una forma completamente determinada?. Debido a esta característica las secuencias generadas son llamadas secuencias de *números pseudoaleatorios.* Este término lo define Lehmer como "una noción vaga que encierra la idea de una sucesión en la cual cada término es impredecible para la persona ajena al problema, cuyos dígitos se someten a cierto número de pruebas, tradicionales a las estadísticas, y dependen en cierta forma del uso que se dará a la sucesión" (Naylor 1971 (Lemher, D. H.)). Bajo esta idea, el carácter de aleatoriedad puede considerarse como dada si la secuencia cumple con cierto número de pruebas estadísticas de aleatoriedad determinadas con anticipación.

Estos métodos superan las ventajas de los anteriores, no existe el problema de dispositivos de entrada o capacidad de almacenamiento y las sucesiones pueden reproducirse, ya que sólo

depende de los pasos aritméticos.

El método mencionado antes, de los *cuadrados centrales*, resulta difícil de analizar, relativamente lento y estadísticamente poco satisfactorio; por lo que fue olvidado y ahora los métodos más utilizados son los de congruencias, desarrollados por Lehmer.

Antes de dar una descripción de los métodos congruenciales para generar números aleatorios, es necesario definir las características con las que debe cumplir un generador de números aleatorios, considerando que se va a programar.

1. El programa debe generar secuencias de números que correspondan a una distribución uniforme y que sean estadísticamente independientes.

2. Debe ser pequeño para que ocupe el mínimo de espacio en memoria.

3. El tiempo de generación de un número debe ser el menor posible, ya que en general se requieren grandes muestras, y además, la producción de números representa sólo una parte del problema.

4. El período del generador debe ser lo más grande posible.

5. Debe permitir reproducir la misma secuencia de números para poder repetir un experimento varias veces. También se debe tener opción para producir secuencias diferentes con el mismo programa[1].

A continuación daremos una descripción de los métodos de congruencias para generar números aleatorios.

2.3.1 Métodos de congruencias lineales

Los métodos de congruencias para generar números aleatorios son completamente determinísticos, debido a que los procesos

[1] Luthe Rodolfo, Olivera Antonio, Schutz Fernando "Métodos Numéricos". México, Limusa, 1978.

aritméticos que se incluyen en los cálculos, determinan unívocamente cada término de la sucesión de números. Aunque estos procesos no son del todo aleatorios, hay fundamentos que nos permiten considerarlos como si en efecto lo fueran, si las sucesiones que resultan se someten consistentemente con éxito a cierto conjunto de pruebas estadísticas diseñadas para probar varias propiedades de los números aleatorios. Por ejemplo, si se puede mostrar que los números en una sucesión aparecen distribuidos en forma uniforme y son estadísticamente independientes, entonces se puede suponer que el proceso es aleatorio pese a su carácter determinístico. Las propiedades (2) y (5) de los requisitos mencionados anteriormente sobre los generadores de números aleatorios se satisfacen automáticamente, al aplicar los métodos de congruencias, debido a que las sucesiones generadas por estos métodos son completamente reproducibles y sólo requieren un mínimo de capacidad de memoria de la computadora. Las propiedades (3) y (4) constituyen los únicos requisitos cuyo grado de satisfacción depende por completo de las propiedades de los métodos aplicados; estas propiedades se analizarán posteriormente.

Los métodos de congruencias se basan en una relación fundamental de congruencia que puede expresarse por medio de la siguiente fórmula recursiva:

$$X_{n+1} = (a\,X_n + c) \bmod m\,, \qquad n \geq 0. \tag{1}$$

donde:

m	es el módulo;	$m > 0$.
a	es un multiplicador;	$0 \leq a < m$.
c	es un incremento;	$0 \leq c < m$.
X_0	es el valor inicial;	$0 \leq X_0 < m$.

La expresión (1) indica que el número aleatorio X_{n+1} es igual al residuo que se obtiene de dividir $(a\,X_n + c)$ entre m. Dicho residuo se puede obtener de la relación

$$X_{n+1} = l - \text{int}\left(\frac{l}{m}\right) * m \qquad (2)$$

donde $l = aX_n + c$, e $\text{int}\left(\frac{l}{m}\right)$ representa la parte entera del cociente l/m.

Los números aleatorios que se producen por este método siempre forman secuencias cíclicas. Esto se debe a que, si el módulo de la división es m, cuando más pueden haber m residuos diferentes: $0,1,\ldots,m-1$ antes de que se repita la secuencia. Por lo tanto la fórmula (1) produce una secuencia de números aleatorios entre 0 y $m-1$. La secuencia de números aleatorios r_n uniformemente distribuidos entre 0 y 1 se obtiene mediante la relación

$$r_n = \frac{X_n}{m} \qquad (3)$$

La máxima cantidad de números aleatorios que se puede generar por el método congruencial antes de que se repita la secuencia se denomina periodo. La longitud del periodo es función de los parámetros a, c, m y X_1 de la expresión (1).

A fin de que la longitud del periodo sea máxima es recomendable seleccionar los parámetros de la fórmula de recurrencia de la siguiente forma[2]:

1) Módulo m.

Es conveniente que el valor de m sea lo más grande posible. El valor más adecuado es:

$$m = p^d$$

[2] Torres Fentanes José A. y Czitrom de Gerez Verónica, "Métodos para la solución de problemas con computadora digital".México, Representaciones y servicios de Ingeniería S.A., 1980.

donde p es el número de dígitos diferentes en el sistema de numeración de la computadora empleada ($p = 2$ para binarias y $p = 10$ para decimales) y d denota el número de dígitos en una computadora.

2) Multiplicador a.

Si se emplean computadoras binarias, emplear

$$a = 8t \pm 3$$

donde t es cualquier entero positivo y a está cercano a $2^{d/2}$. Si se emplean computadoras decimales emplear:

$$a = 200t + k$$

donde t es cualquier entero positivo, k es cualquier número primo menor a 200 y diferente de 1; y a esta cercano a $2^{d/2}$.

3) Incremento c.

Se puede hacer

$$c = 0$$

sin deteriorar apreciablemente el método congruencial lineal. Esto reduce en uno el número de operaciones en cada iteración, con lo que se disminuye el tiempo de cálculo. Cuando la fórmula (1) se utiliza con c=0 el método se denomina *congruencial multiplicativo* y cuando $c \neq 0$ se denomina *congruencial mixto*.

4) Valor inicial X_0.

Para computadoras binarias el valor inicial puede ser cualquier entero impar, y para computadoras decimales deberá ser un entero impar no múltiplo de 5.

Seleccionando los parámetros en la forma indicada se puede

demostrar que la longitud del máximo periodo obtenible es de $m/4$ para computadoras binarias y $m/20$ para computadoras decimales [3].

Se han desarrollado tres métodos básicos de congruencias para generar números aleatorios mediante el empleo de distintas versiones de la relación dada en la ecuación (1). El objetivo de cada uno de los métodos es la generación, en un mínimo de tiempo, de sucesiones con un periodo máximo. Estos métodos de congruencias: el aditivo, el multiplicativo y el mixto.

A. Método aditivo de congruencias

Este método presupone k valores iniciales dados, con k un entero positivo y calcula una sucesión de números mediante la siguiente relación de congruencia:

$$x_{i+1} \equiv (x_i + x_{i-k}) \bmod m \qquad (4)$$

Si k=1, la ecuación (4) genera la bien conocida sucesión de Fibonacci, la cual se comporta como las sucesiones que se pueden obtener por el método multiplicativo de congruencias, en las que se tiene el muy desfavorable multiplicador $a = (1 + \sqrt{5})/2$; las propiedades estadísticas de la sucesión tienden a mejorarse a medida que k se incrementa. Este es el único método que produce períodos mayores que m. El proceso aditivo, un tanto simple para generar números aleatorios de acuerdo con la fórmula (4), ha sido programado en varias computadoras de acuerdo con las modificaciones que se han considerado en la definición de parámetros y que han sido aprobadas por Green. La fórmula básica empleada es

$$n_j \equiv (n_{j-1} + n_{j-k}) (\text{mod } 2^b),$$

[3] Torres y Czitrom, op. cit.

donde b es el número de términos binarios acarreados por una computadora binaria. Al utilizar este generador aditivo se deben proveer k números aleatorios al almacenamiento original. Los números aleatorios generados de esta manera tienen un periodo igual a $p_k.2^{b-1}$ en donde p_k es una constante que depende tanto de k como de b. Aún no se han reportado en la literatura pruebas comparativas en relación a las ventajas del método aditivo sobre los métodos multiplicativos. Las características relativas a la velocidad de los generadores aditivos dependen de los códigos de programación y las computadoras utilizadas; las propiedades estadísticas de los números aleatorios requieren pruebas empíricas al igual que cualquier otro método.

B. Método multiplicativo de congruencias

Este método calcula una sucesión $\{n_i\}$ de enteros no negativos, cada uno de los cuales siempre es menor que m, por medio de la relación de congruencia

$$x_{i+1} \equiv a\ x_i \pmod{m}$$

Este método es un caso especial de la citada relación de congruencia (ecuación 1), en donde $c = 0$. Se ha encontrado que el método multiplicativo se comporta de manera muy satisfactoria en lo que toca a su estadística; esto es, tanto las pruebas de frecuencia y las de serie, como otras pruebas relativas a la aleatoriedad, cuando se aplican a las sucesiones que se obtienen mediante este método indican que los números aleatorios así encontrados están uniformemente distribuidos y no correlacionados. Más aún, para fines de asegurar un período máximo en las sucesiones generadas por este método, es posible imponer condiciones convenientes tanto para el multiplicador a como para el valor inicial Xo con que se empieza el cálculo. Existen ventajas que también ofrece el método multiplicativo en términos de las velocidades de cálculo.

C. Método mixto de congruencias

Los números que se obtienen mediante la relación de congruencia (ecuación 1) en su forma original (con *a* y *c* mayores que cero), se dice que son generados por el *método mixto de congruencias*. Tal método ofrece algunas pequeñas ventajas, al compararlo con el método multiplicativo, en términos de incremento en la velocidad de cálculo y pérdida de periodicidad en los últimos dígitos. La ventaja principal del método mixto radica en su periodo completo. Aunque su comportamiento estadístico es generalmente bueno, en pocos casos resulta inaceptable por completo.

CAPITULO 3

PRUEBAS ESTADISTICAS PARA LOS NUMEROS ALEATORIOS

Las propiedades estadísticas de los números aleatorios generados por los métodos que se han delineado en el capítulo anterior, deben coincidir con las propiedades estadísticas de los generados por un instrumento aleatorio idealizado, que selecciona los números dentro de un intervalo unitario (0,1) en forma independiente y de igual manera para todos ellos. En este sentido, es claro que los números aleatorios obtenidos mediante programas de computación no son totalmente aleatorios, lo cual se debe a que tales números están completamente determinados por los datos iniciales y tienen una precisión limitada. Sin embargo, en la medida en que nuestros números pseudoaleatorios puedan pasar las estadísticas, denotados por el instrumento aleatorio idealizado, estos números pseudoaleatorios pueden tratarse como verdaderos números aleatorios aunque no lo sean.

3.1 PROCEDIMIENTOS DE PRUEBAS GENERALES

3.1.1 Prueba "Chi-cuadrada"

La prueba chi-cuadrada (X^2) es tal vez la mejor conocida de todas las pruebas estadísticas, y esta tiene un método básico que se usa junto con muchas otras pruebas.

Se hace un número n suficientemente grande de observaciones. Se cuenta el número de observaciones que caen en cada una de las k categorías y calculamos la cantidad V dada en las ecuaciones (1) y

(2) dadas a continuación

$$V = \sum_{1 \le s \le k} \frac{(Y_s - np_s)^2}{np_s} \qquad (1)$$

$$V = \frac{1}{n} \sum_{1 \le s \le k} \left(\frac{Y_s^2}{p_s}\right) - n \qquad (2)$$

Después V se compara con los números de la tabla 3.1, con $v = k-1$. Si V es menor que la entrada 1% o mayor que la entrada 99%, rechazamos los números como no suficientemente aleatorios. Si V está entre las entradas 1% y 5% o entre las entradas 95% y 99% , los números son "sospechosos"; si (por interpolación de la tabla) V está entre las entradas de 5% y 10% o las entradas 90% y 95%, los números serán "casi sospechosos". La prueba chi-cuadrada con frecuencia se aplica por lo menos tres veces a diferentes conjuntos de datos, y si al menos dos de los tres resultados son sospechosos los números se consideran como no suficientemente aleatorios (Knuth,1969).

3.1.2 La prueba Kolgomorov-Smirnov

Como hemos visto, la prueba chi-cuadrada se aplica en situaciones donde las observaciones pueden caer en un número finito de categorías. Sin embargo, no es raro, considerar cantidades aleatorias que pueden asumir valores infinitamente. Por ejemplo, un número real aleatorio entre cero y uno puede tomar valores infinitamente; aunque sólo un número finito de estos puede representarse en la computadora, nosotros deseamos que nuestros valores aleatorios se comporten esencialmente como si fueran números realmente aleatorios.

	$P = 1\%$	$P = 5\%$	$P = 25\%$	$P = 50\%$	$P = 75\%$	$P = 95\%$	$P = 99\%$
$\nu = 1$	0.00016	0.00393	0.1015	0.4549	1.323	3.841	6.635
$\nu = 2$	0.02010	0.1026	0.5753	1.386	2.773	5.991	9.210
$\nu = 3$	0.1148	0.3518	1.213	2.366	4.108	7.815	11.34
$\nu = 4$	0.2971	0.7107	1.923	3.357	5.385	9.488	13.28
$\nu = 5$	0.5543	1.1455	2.675	4.351	6.626	11.07	15.09
$\nu = 6$	0.8720	1.635	3.455	5.348	7.841	12.59	16.81
$\nu = 7$	1.239	2.167	4.255	6.346	9.037	14.07	18.48
$\nu = 8$	1.646	2.733	5.071	7.344	10.22	15.51	20.09
$\nu = 9$	2.088	3.325	5.899	8.343	11.39	16.92	21.67
$\nu = 10$	2.558	3.940	6.737	9.342	12.55	18.31	23.21
$\nu = 11$	3.053	4.575	7.584	10.34	13.70	19.68	24.73
$\nu = 12$	3.571	5.226	8.438	11.34	14.84	21.03	26.22
$\nu = 15$	5.229	7.261	11.04	14.34	18.25	25.00	30.58
$\nu = 20$	8.260	10.85	15.45	19.34	23.83	31.41	37.57
$\nu = 30$	14.95	18.49	24.48	29.34	34.80	43.77	50.89
$\nu = 50$	29.71	34.76	42.94	49.33	56.33	67.50	76.15
$\nu > 30$	$\nu + \sqrt{2\nu}x_p + 2/3\, x_p^2 - 2/3 + O(1/\sqrt{\nu})$						
$x_p =$	-2.33	-1.64	-0.675	0.00	0.675	1.64	2.33

TABLA 3.1

Puntos porcentuales seleccionados de la distribución Chi-cuadrada.

Comúnmente se usa una notación general para especificar las distribuciones de probabilidad, si ellas son finitas o infinitas, en los estudios de probabilidad y estadística. Suponiendo que deseamos especificar la distribución de los valores de una cantidad aleatoria X; hacemos esto en términos de la función de distribución $F(x)$, donde

$$F(x) = \text{la probabilidad de que } (X \leq x).$$

En la figura 3.1 se muestran tres ejemplos. Primero vemos la función de distribución para un bit aleatorio, es decir, para el caso donde X toma sólo dos valores 0 y 1, cada uno con probabilidad de 1/2. La parte (b) de la figura muestra una función de distribución para un *número real aleatorio* entre cero y uno, así la probabilidad de que $X \leq x$ es simplemente igual a x cuando $0 \leq x \leq 1$; por ejemplo, la probabilidad de que $X \leq \frac{2}{3}$ es, naturalmente, $\frac{2}{3}$. Y la parte (c) muestra la distribución del valor V en la prueba chi-cuadrada (mostrada aquí con 10 grados de libertad); esta es una distribución que ya hemos visto representada en otra forma en la tabla 3.1. Observemos que $F(x)$ siempre se incrementa desde 0 hasta 1 conforme x se incrementa desde $-\infty$ hasta $+\infty$.

Si tomamos n observaciones independientes de la cantidad aleatoria X, obtenemos así los valores $X_1, X_2, \ldots, X_n$, podemos formar la función de distribución empírica $F_n(x)$, donde

$$F_n(x) = \frac{\text{números de } X_1, X_2, \ldots, X_n \text{ que son} \leq x}{n} \qquad (3)$$

La figura 3.2 ilustra tres funciones de distribución empíricas (mostradas como lineas en zigzag, aunque estrictamente hablando las líneas verticales no son parte de la gráfica de $F_n(x)$), superpuestas en una gráfica de la función de distribución actual supuesta $F(x)$. Conforme n es grande, $F_n(x)$ será mejor y más aproximada a $F(x)$.

La prueba Kolgomorov-Smirnov (prueba KS) puede usarse cuando

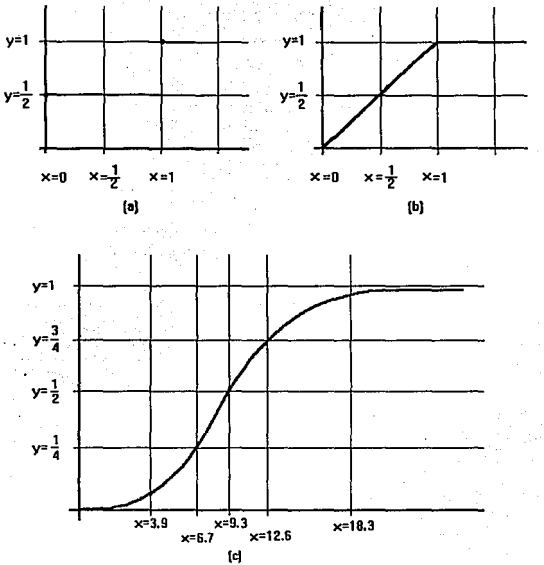


FIGURA 3.1

Ejemplos de funciones de distribución.

$F(x)$ no tiene saltos. Esto se basa en la diferencial entre $F(x)$ y $F_n(x)$. Una fuente deficiente de números aleatorios dará funciones de distribución empíricas que no se aproximan lo suficientemente bien a $F(x)$. La figura 3.2(b) muestra un ejemplo en el que las X_i están demasiado altas, así que la función de distribución empírica está demasiado baja. La parte (c) muestra un ejemplo aún peor; es claro que tales desviaciones tan grandes entre $F_n(x)$ y $F(x)$ son muy improbables, y la prueba KS se usa para decirnos cuán improbable es.

Para hacer esta prueba, formamos los siguientes estadísticos:

$$K_n^+ = \sqrt{n} \max_{-\infty < x < +\infty} (F_n(x) - F(x)) ;$$
$$K_n^- = \sqrt{n} \max_{-\infty < x < +\infty} (F(x) - F_n(x)) ; \qquad (4)$$

Aquí K_n^+ mide la máxima desviación en el caso de que F_n sea mayor que F, y K_n^- mide la máxima desviación cuando F_n es menor que F. Los estadísticos para los ejemplos de la figura 3.2 son

	(a)	(b)	(c)
K_{20}^+	0.492	0.134	0.313
K_{20}^-	0.536	1.027	2.101

Nota: La desviación estandar de Fn(x) es proporcional a $1/\sqrt{n}$; entonces el factor $\sqrt{n}$ amplia los estadísticos K_n^- , K_n^+ en forma tal que esta desviación estandar es independiente de n.

Como en la prueba chi-cuadrada, podemos ahora observar los valores K_n^+, K_n^- en una tabla "percentil" para determinar si son baja o altamente significativos. Puede usarse la tabla 3.2 para este propósito, con K_n^+ y K_n^-. Por ejemplo, hay un 75% de probabilidad de que K_{20}^- sea 0.7975 o menor. A diferencia de la prueba chi-cuadrada, las entradas de la tabla *no* son meras aproximaciones que se mantienen para valores grandes de n; la tabla 3.2 da valores exactos (con excepción de errores de

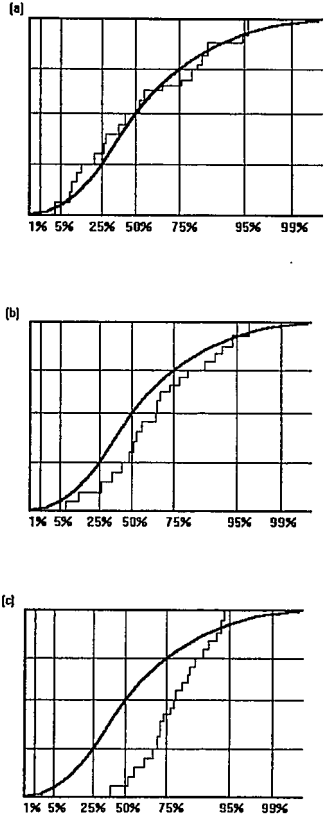


FIGURA 3.2

Ejemplos de distribuciones empíricas

redondeo), y la prueba KS puede usarse de manera formal para cualquier valor de n.

Así como están, las fórmulas (4) no son fácilmemte adaptables para cálculos por computadora, puesto que nos preguntamos sobre un máximo de la infinidad de valores de x. Sin embargo, del hecho de que $F(x)$ está aumentando y el hecho de que $F_n(x)$ aumenta sólo en pasos finitos, podemos derivar un procedimiento sencillo para evaluar los estadísticos K_n^+ y K_n^-:

Paso 1. Obtener las observaciones $X_1, X_2, \ldots, X_n$.

Paso 2. Rearreglar las observaciones en orden ascendente, es decir, en forma que $X_1 \le X_2 \le \ldots \le X_n$.

Paso 3. Los estadísticos deseados ahora están dados por las fórmulas

$$K_n^+ = \sqrt{n}\ \max_{1 \le j \le n} \left(\frac{j}{n} - F(x_j) \right);$$
$$K_n^- = \sqrt{n}\ \max_{1 \le j \le n} \left(F(X_j) - \frac{j-1}{n} \right). \qquad (5)$$

En una elección apropiada del número de observaciones, n, se facilita más la aplicación de esta prueba, en lugar de la prueba X^2, aunque algunas consideraciones son similares. Si las variables aleatorias X_j actualmente corresponden a una función de distribución $G(x)$, mientras se consideraron con una función de probabilidad $F(x)$, esta tomará grandes valores de n rechazando la hipótesis de que $G(x)=F(x)$; necesitamos n bastante grande para que las distribuciones empíricas $G_n(x)$ y $F(x)$ esperadas sean observablemente diferentes. De otra manera, grandes valores de n tenderán a un promedio fuera del comportamiento no aleatorio local, y tal comportamiento tiene características indeseables que tienen importancia significante en la mayoría de aplicaciones computacionales de números aleatorios; esto crea un caso para valores pequeños de n. Un buen acuerdo debería tomar n igual a 1000, por decir, y tomar un número grande adecuado de cálculos de K_{1000}^+ en diferentes partes de una secuencia aleatoria, así

	p=1%	p=5%	p=25%	p=50%	p=75%	p=95%	p=99%
n = 1	0.01000	0.05000	0.2500	0.5000	0.7500	0.9500	0.9900
n = 2	0.01400	0.06749	0.2929	0.5179	0.7071	1.0980	1.2728
n = 3	0.01699	0.07919	0.3112	0.5147	0.7539	1.1017	1.3589
n = 4	0.01943	0.08789	0.3202	0.5110	0.7642	1.1304	1.3777
n = 5	0.02152	0.09471	0..3249	0.5245	0.7674	1.1392	1.4024
n = 6	0.02336	0.1002	0.3272	0.5319	0.7703	1.1463	1.4144
n = 7	0.02501	0.1048	0.3280	0.5364	0.7755	1.1537	1.4246
n = 8	0.02650	0.1086	0.3280	0.5392	0.7797	1.1586	1.4327
n = 9	0.02786	0.1119	0.3274	0.5411	0.7825	1.1624	1.4388
n = 10	0.02912	0.1147	0.3297	0.5426	0.7845	1.1658	1.4440
n = 11	0.03028	0.1172	0.3330	0.5439	0.7863	1.1688	1.4484
n = 12	0.03137	0.1193	0.3357	0.5453	0.7880	1.1714	1.4521
n = 15	0.03424	0.1244	0.3412	0.5500	0.7926	1.1773	1.4606
n = 20	0.03807	0.1298	0.3461	0.55475	0.7975	1.1839	1.4698
n = 30	0.04354	0.1351	0.3509	0.5605	0.8036	1.1916	1.4801
n > 30	y_p - $1/(6\sqrt{n}) + O(1/n)$, donde $y^2_p = 1/2\ln(1/(1-p))$						
	0.07089	0.1601	0.3793	0.5887	0.8326	1.2239	1.5174

TABLA 3.2

Puntos porcentuales seleccionados de la distribuciòn K^+n y K^-n

obtenemos valores

$$K\overset{+}{1000}(1), \qquad K\overset{+}{1000}(2), \qquad \ldots, \qquad K\overset{+}{1000}(r). \tag{6}$$

Podemos aplicar de nuevo la prueba KS para estos resultados: Asignar ahora $F(x)$ como la función de distribución para $K\overset{+}{1000}$, y determinar la distribución empírica $F_r(x)$ obtenida de los valores observados en (6). Afortunadamente, la función $F(x)$ en este caso es muy simple ; para un valor gande de n como n=1000, la distribución de K_n^+ está aproximada por

$$F_\infty(x) = 1 - e^{-2x^2} \qquad x \geq 0. \tag{7}$$

El mismo comentario se aplica a K_n^-, puesto que K_n^+ y K_n^- tienen el mismo comportamiento esperado. Este método de usar varias pruebas para n moderadamente grande, y combinando las observaciones más tarde en otra prueba KS, tenderá a detectar el comportamiento no aleatorio local y el global (Knuth, 1969).

3.2 PRINCIPALES PRUEBAS DE ALEATORIEDAD

En esta sección discutiremos las pruebas específicas que han sido aplicadas a secuencias para investigar su aleatoriedad.

Cada prueba se aplica a la secuencia

$$< U_n > = U_0, U_1, U_2, \ldots \tag{1}$$

de números reales, los cuales pretenden ser independientes y uniformemente distribuidos entre 0 y 1. Algunas de estas pruebas estan diseñadas primeramente para secuencias de valores enteros, en lugar de secuencias de valores reales. En este caso, la secuencia auxiliar:

$$< Y_n > = Y_0, Y_1, Y_2, \ldots \qquad (2)$$

la cuàl se define por la regla

$$Y_n = \lfloor dU_n \rfloor \qquad (3)$$

Esta es una secuencia de enteros que pretende ser independientes y uniformemente distribuidos entre 0 y d-1. El número d se elige por conveniencia; por ejemplo, tendríamos $d = 64 = 2^6$ en una computadora binaria, así Y_n representa los 6 dígitos más significativos de la representación binaria de U_n. El valor de d debería ser bastante grande de manera que la prueba sea significativa, pero no tan grande, debido a que la prueba sería difícil de aplicar.

Las cantidades U_n, Y_n y d tendrán el significado anterior para toda esta sección, aunque el valor de d probablemente será diferente en diferentes pruebas.

A. Prueba de frecuencias.

El primer requisito que debe cumplir la secuencia (1) es que los números sean, en efecto, uniformemente distribuidos entre cero y uno.. Hay dos formas de hacer esta prueba: (a) Usando la prueba Kolgomorov-Smirnov, con $F(x) = x$ para $0 \le x \le 1$. (b) Asignando a d un valor adecuado, por ejemplo, 100 en una computadora decimal, 64 o 128 en una computadora binaria, y usar la secuencia (2) en lugar de la (1). Para cada entero r , $0 \le r < d$, contar el número de veces que $Y_j = r$ para $0 \le j < n$, y aplicar la prueba Chi-cuadrada usando k=d y probabilidad $p_s = 1/d$ para cada categoría.

B. Prueba de series.

Con frecuencia necesitamos parejas de números sucesivos que estén uniformemente distribuidos de una manera independiente.

Para llevar a cabo la prueba de series, simplemente contamos el número de veces que ocurre la pareja $(Y_{2j},Y_{2j+1}) = (q,r)$, para $0 \le j < n$; esto, se hace para cada pareja de enteros (q,r) con $0 \le q, r < d$, y se aplica la prueba chi-cuadrada a estas $k = d^2$ categorías con probabilidad $1/d^2$ en cada categoría. Como con la prueba de frecuencias, d puede ser cualquier número adecuado, pero será algo más pequeño que los valores sugeridos antes, donde una prueba chi-cuadrada válida tendrá n grande comparado con k (es decir, al menos $n \ge 5d^2$).

Claramente podemos generalizar esta prueba para triples, cuadruples, etc., en lugar de pares; sin embargo, el valor de d debe disminuirse para evitar tener demasiadas categorías.

Observemos que $2n$ números de la secuencia (2) son usados para esta prueba para hacer n observaciones. Sería un error aplicar la prueba de series en las parejas (Y_0,Y_1), (Y_1,Y_2), ..., (Y_{n-1},Y_n); podríamos aplicar otra prueba serial en los pares (Y_{2j+1},Y_{2j+2}), y esperar que la secuencia pase ambas pruebas. Alternativamente, I. J. Good probó que si d es primo, y si se usan las parejas (Y_0,Y_1), (Y_1,Y_2), ..., (Y_{n-1},Y_n), y si usamos el método usual chi-cuadrada para calcular las estadísticas V_2 para la prueba de series y V_1 para la prueba de frecuencias en Y_0, ..., Y_{n-1} con el mismo valor de d, entonces $V_2 - 2V_1$ tendría la distribución chi-cuadrada con $(d-1)^2$ grados de libertad cuando n es grande.

C. Prueba de intervalos.

Esta prueba se usa para determinar la longitud de "intervalos" entre ocurrencias de U_j en un cierto rango. Si α y β son dos números reales con $0 \le \alpha < \beta \le 1$, deseamos considerar las longitudes de las subsecuencias consecutivas U_j, U_{j+1}, ..., U_{j+r} en las que U_{j+r} está entre α y β pero las otras U's no. (Esta

subsecuencia de $r + 1$ números representa un intervalo de longitud r).

Algoritmo G.

El siguiente algoritmo, aplicado a la secuencia (1) para valores de α y β cualesquiera, cuenta el número de intervalos de longitudes 0, 1, ..., $t-1$ y el número de intervalos de longitud $\geq t$, hasta que se han tabulado n intervalos.

G1. Iniciar $j = -1$, $s = 0$, e iniciar COUNT[r] = 0 para $0 \leq r \leq t$.

G2. Iniciar $r = 0$.

G3. Incrementar j en 1. Si $U_j \geq \alpha$ y $U_j < \beta$, ir al paso G5.

G4. Incrementar r en 1, y regresar al paso G3.

G5. (Se ha encontrado un intervalo de longitud r)
Si $r \geq t$, incrementar COUNT[t] en 1, en caso contrario incrementar COUNT[r] en 1.

G6. Incrementar s en 1. Si $s < n$, regresar al paso G2.

Después de que se ha ejecutado este algoritmo, se aplica la prueba chi-cuadrada a los valores k = $t + 1$ de COUNT[0], COUNT[1], ..., COUNT[t], usando las siguientes probabilidades:

$$p_0 = p, \quad p_1 = p(1-p), \; p_2 = p(1-p)^2, \quad \ldots, \quad p_{t-1} = p(1-p)^{t-1}, \quad p_t = (1-p)^t. \tag{4}$$

Aquí $p = \beta - \alpha$, la probabilidad de que $\alpha \leq U_j < \beta$. Los valores de n y t se eligen en la forma usual, así que cada uno de los valores de COUNT[r] se espera que sea 5 o más, de preferencia más.

La prueba de intervalos con frecuencia se aplica con $\alpha = 0$ o $\beta = 1$ para suprimir una de las comparaciones del paso G3. Los casos especiales $(\alpha, \beta) = (0, \frac{1}{2})$ o $(\frac{1}{2}, 1)$ son las pruebas que con frecuencia se denominan como "corridas arriba de la media" y "corridas abajo de la media" respectivamente.

Observamos que la prueba de intervalos como se ha descrito

arriba identifica las longitudes de n intervalos, no identifica las longitudes de intervalos entre n *números*. Si la secuencia $\langle U_n \rangle$ es suficientemente no aleatoria, el algoritmo G no tendrá fin.

D. Prueba de Póker

La prueba de póker "clásica" considera n grupos de cinco enteros sucesivos, $(Y_{sj}, Y_{sj+1}, \ldots, Y_{sj+4})$ para $0 \le j < n$, e identifica cuáles de los siguientes patrones presenta cada quintuple:

Todos diferentes:	*abcde*	Casa llena:	*aaabb*
Un par:	*aabcd*	Cuatro de un tipo:	*aaaab*
Dos pares:	*aabbc*	Cinco de un tipo:	*aaaaa*
Tres de un tipo:	*aaabc*		

Una prueba chi-cuadrada se basa en el número de quintuples en cada categoría.

Es razonable preguntarse por alguna versión más sencilla de esta prueba, para facilitar la programación involucrada. Un buen acuerdo podría ser simplemente contar el número de valores *distintos* en el conjunto de cinco. Tendríamos entonces cinco categorías:

5 diferentes = todos diferentes;
4 diferentes = un par;
3 diferentes = dos pares, o tres de un tipo;
2 diferentes = casa llena, o cuatro de un tipo;
1 diferente = cinco de un tipo.

Esta clasificación es más fácil de determinar, y la prueba es aproximadamente igual de eficiente.

En general podemos considerar n grupos de k números sucesivos, y podemos contar el número de k-tuples con r valores diferentes. Entonces se aplica una prueba chi-cuadrada, usando la probabilidad

$$p_r = \frac{d(d-1)\ \ldots\ (d-r+1)}{d^k} \left\{ {k \atop r} \right\} \qquad (5)$$

Puesto que la probabilidad p_r es muy pequeña cuando $r=1$ o 2, generalmente agrupamos algunas categorías de baja probabilidad antes de aplicar la prueba chi-cuadrada.

Para deducir la fórmula apropiada para p_r, debemos contar cuántos de los d^k k-tuples de números entre 0 y $d-1$ tienen exactamente r elementos diferentes, y dividir el total por d^k. Puesto que $d(d-1)\ldots(d-r+1)$ es el número de opciones de ordenación de r cosas de un conjunto de d objetos, sólo necesitamos mostrar que $\left\{ {k \atop r} \right\}$ es el número de formas de dividir exactamente un conjunto de k elementos en r partes.

E. Prueba del colector de cupones

Esta prueba está relacionada a la prueba del póker así como la prueba de intervalos se relaciona con la prueba de frecuencias. Se usa la secuencia $Y_0, Y_1, \ldots$, e identificamos la longitud de los segmentos $Y_{j+1}, Y_{j+2}, \ldots, Y_{j+r}$ necesarios para obtener un "conjunto completo" de enteros desde 0 hasta $d-1$. El algoritmo C describe este proceso:

Algoritmo C.

Dada una secuencia de enteros $Y_0, Y_1, \ldots$, con $0 \le Y_j < d$, este algoritmo cuenta las longitudes de n segmentos "colectores de cupón" consecutivos. Al finalizar este algoritmo, COUNT[r] es el número de segmentos con longitud r, para $d \le r < t$, y COUNT[t] es el número de segmentos con longitud $\ge t$.

C1. Iniciar $j = -1$, $s = 0$, y COUNT[r] = 0 para $d \le r \le t$.

C2. Iniciar $q = r = 0$, y OCCURS[k] = 0 para $0 \le k < d$.

C3. Incrementar r y j en 1. Si OCCURS[Y_j] $\neq$ 0, repetir este paso.

C4. Iniciar OCCURS[Y_j] = 1 y $q = q + 1$. (La subsecuencia observada hasta ahora contiene q valores distintos; si

$q = d$, tenemos un conjunto completo), Si $q < d$, regresar al paso C3.

C5. Si $r \geq t$, incrementar COUNT[t] en 1, en otro caso incrementar COUNT[r] en 1.

C6. Incrementar s en 1. Si $s < n$, regresar al paso C2.

Como ejemplo de este algoritmo podríamos pensar en un niño que colecciona d tipos de cupones, los cuales están aleatoriamente distribuidos en las cajas de cereales; el permanecería comiendo más cereal hasta tener un cupón de cada tipo.

Se aplica una prueba chi-cuadrada a COUNT[d], COUNT[$d + 1$], ..., COUNT[t], con $k = t - d + 1$, después de que el algoritmo ha contado n longitudes. Las probabilidades correspondientes son

$$p_r = \frac{d!}{d^r} \left\{ \begin{matrix} r-1 \\ d-1 \end{matrix} \right\}, \quad d \leq r < t; \qquad p_r = 1 - \frac{d!}{d^{t-1}} \left\{ \begin{matrix} t-1 \\ d \end{matrix} \right\}. \qquad (6)$$

Para deducir estas probabilidades, simplemente notemos que si q_r denota las probabilidades de que una subsecuencia de longitud r está incompleta, entonces

$$q_r = 1 - \frac{d!}{d^r} \left\{ \begin{matrix} r \\ d \end{matrix} \right\}$$

por la ecuación(5); esto significa que tenemos una r-tuple de elementos que no tienen todos los d valores diferentes. Entonces (6) se sigue de las relaciones $p_r = q_{r-1} - q_r$ para $d \leq r < t$; $p_t = q_{t-1}$.

F. Prueba de permutaciones.

Divide la secuencia inicial en n grupos de t elementos cada uno, esto es, (U_{jt}, U_{jt+1}, ..., U_{jt+t-1}) para $0 \leq j < n$. Los elementos en cada grupo pueden tener $t!$ posibles ordenaciones relativas; se cuenta el número de veces que sucede cada ordenación, y se plica una prueba chi-cuadrada con $k = t!$ y con

probabilidad $1/t!$ para cada ordenación.

Por ejemplo, si $t = 3$ tendríamos seis posibles categorías, de acuerdo a si $U_{3j} < U_{3j+1} < U_{3j+2}$ o $U_{3j} < U_{3j+2} < U_{3j+1}$ o ... o $U_{3j+2} < U_{3j+1} < U_{3j}$. En esta prueba consideramos que la igualdad entre U's no ocurre; por lo que la probabilidad de que dos U's sean iguales es cero.

Una forma conveniente de realizar la prueba de permutaciones en una computadora es por el siguiente algoritmo:

Algoritmo P.

Dada una secuencia de distintos elementos $(U_1, \ldots, U_t)$, calculamos un entero $f(U_1, \ldots, U_t)$ tal que

$$0 \le f(U_1, \ldots, U_t) < t!,$$

y $f(U_1, \ldots, U_t) = f(V_1, \ldots, V_t)$ si y solo si $(U_1, \ldots, U_t)$ y $(V_1, \ldots, V_t)$ tienen el mismo orden relativo.

P1. Iniciar $r = t$, $f = 0$. (En este algoritmo tendremos $0 \le f < t!/r!$).

P2. Encontrar el máximo de $\{U_1, \ldots, U_t\}$, y suponer que U_s es el máximo. Iniciar $f = r*f + s - 1$.

P3. Intercambiar $U_r \leftrightarrow U_s$.

P4. disminuir r en 1. Si $r > 1$, regresar al paso P2.

Observemos que la secuencia $(U_1, \ldots, U_t)$ habrá sido ordenado en forma ascendente cuando finalice este algoritmo. Para probar que el resultado f unicamente caracteriza el orden *inicial* de $(U_1, \ldots, U_t)$, observando que el algoritmo P puede ser recorrido hacia atrás: Para $r = 2, 3, \ldots, t$, iniciar $s = f \bmod r$, $f = \lfloor f/r \rfloor$, e intercambiando U_rU_s. Es fácil ver que esto anulará los efectos de los pasos P2-P4; por lo que ningún par de permutaciones puede producir el mismo valor de f, y el algoritmo P se ejecuta de la manera expuesta.

La idea principal que subraya el algoritmo P es una representación radix-mixta llamada el "sistema del número factorial": Cada entero en el rango $0 \le f < t!$ puede escribirse únicamente en la forma

$$f = (\cdots(c_{t-1} \times (t-1) + c_{t-2}) \times (t-2) + \cdots + c_2) \times 2 + c_1$$
$$= (t-1)!c_{t-1} + (t-2)!c_{t-2} + \cdots + 2!c_2 + 1!c_1 \qquad (7)$$

donde los "dígitos" c_j son enteros que satisfacen

$$0 \le c_j \le j, \qquad \text{para } 1 \le j < t. \qquad (8)$$

En el algoritmo P, $c_{r-1} = s - 1$ cuando P2 se ejecuta para un valor de r dado.

G. Prueba de corridas.

La naturaleza de la aleatoriedad oscilante de las secuencias de los números aleatorios se puede comprobar mediante las llamadas pruebas de corridas. Se conocen dos tipos de estas pruebas: las pruebas para corridas arriba y abajo de la media (tratadas anteriormente) y las de arriba y abajo que se tratan a continuación.

Para una secuencia de n números aleatorios U_1, U_2, ..., U_n definimos una sucesión binaria S de $n-1$ bits, cuyo i-ésimo término es igual a *cero*, si $U_i < U_{i+1}$ es igual a *uno* si $U_i > U_{i+1}$. Una subsecuencia de k ceros, enmarcada por unos en cada extremo, recibe el nombre de corrida de ceros de longitud k; similarmente se definen las corridas de unos. La prueba consiste en determinar las ocurrencias de corridas de distinta longitud y comparar estos conteos con sus valores teóricos correspondientes esperados. Dichos valores basados en una muestra "verdaderamente" aleatoria son:

$$\frac{(2n-1)}{3} \qquad \text{para el número total de corridas;}$$

$$\frac{(5n + 1)}{12}$$ para corridas de longitud 1;

$$\frac{(11n - 14)}{60}$$ para corridas de longitud 2;

.....

$$\frac{2[(k^2 + 3k + 1)n - (k^3 + 3k^2 - k - 4)]}{(k + 3)!}$$

para corridas de longitud k con $k < n - 1$,

$$\frac{2}{n!}$$ para corridas de longitud $n - 1$.

Nuevamente, la confiabilidad del ajuste se prueba con el criterio de la chi-cuadrada, a fin de comprobar si un generador de números aleatorios resulta aceptable a un cierto nivel de significancia. Una característica común de las sucesiones de números no aleatorios es la de tener un exceso de corridas muy largas.

H. Prueba máximo de t.

Para $0 \leq j < n$, sea $V_j = \max(U_{tj}, U_{tj+1}, \ldots, U_{tj+t-1})$. Ahora aplicamos la prueba Kolgomorov-Smirnov a la secuencia $V_0, V_1, \ldots, V_{n-1}$, con la función de distribución $F(x) = x^t$, $0 \leq x \leq 1$. Alternativamente, aplicamos la prueba de frecuencias a la secuencia $V_0^t, V_1^t, \ldots, V_{n-1}^t$.

Para verificar esta prueba, debemos mostrar que la función de distribución $F(x) = x^t$. La probabilidad de que $\max(U_1, \ldots, U_t) \leq x$ es la probabilidad de que $U_1 \leq x$ y $U_2 \leq x$ y $\cdots$ y $U_t \leq x$, el cual es el producto de las probabilidades individuales, denotado $xx\ldots x = x^t$.

I. Prueba de colisión.

La prueba chi-cuadrada puede aplicarse sólo cuando se espera un número no trivial de artículos esperados en cada categoría. Pero existe otro tipo de prueba que puede usarse cuando el número de categorías es mucho más grande que el número de observaciones.

Supongamos que tenemos m urnas y lanzamos n bolas al azar en estas urnas, donde m es mucho mayor a n. La mayoría de las bolas caerán en urnas que estaban vacías, pero si una bola cae en una urna que ya contenía al menos una bola decimos que ha ocurrido una "colisión". La prueba de colisión cuenta el número de colisiones, y un generador pasa esta prueba si no induce demasiadas o muy pocas colisiones.

Para entender esto, supongamos $m = 2^{20}$ y $n = 2^{14}$. Entonces cada urna recibirá una 64^{va} parte de una bola, en promedio. La probabilidad de que una urna dada contenga exactamente k bolas es $p_k = \binom{n}{k} m^{-k}(1-m^{-1})^{n-k}$ así el número esperado de colisiones por urna es $\sum_{k\geq 1}(K-1)p_k = \sum_{k\geq 0} k p_k - \sum_{k\geq 1} p_k = n/m - 1 + p_0$. Dado que $p_0 = (1 - m^{-1})^n = 1 - n/m + \binom{n}{2} m^{-2} +$ terminos más pequeños, encontramos que el promedio del número total de colisiones tomando aproximadamente el total m de urnas es ligeramente menor a $n^2/2m = 128$.

La prueba de colisiones puede utilizarse para estimar un generador de números aleatorios en un amplio número de dimensiones. Por ejemplo, cuando $m = 2^{20}$ y $n = 2^{14}$ podemos probar la aleatoriedad 20-dimensional de un generador aleatorio designando $d = 2$ y creando vectores 20-dimensionales $V_j = (Y_{20j}, Y_{20j+1}, \ldots, Y_{20j+19})$ para $0 \leq j < n$. Basta mantener una tabla de $m = 2^{20}$ bits para determinar las colisiones, un bit para cada posible valor del vector V_j; en una computadora con 32 bits por palabra, esto asciende a 2^{15} palabras. Inicialmente todos los 2^{20} bits de esta tabla son puestos en cero; entonces para cada V_j, si el bit correspondiente ya es 1 registramos una colisión, en otro caso iniciamos el bit a 1. Esta prueba puede usarse también en 10 dimensiones con $d = 4$, y así sucesivamente.

Para decidir si pasa la prueba, podemos utilizar la siguiente tabla de porcentajes cuando $m = 2^{20}$ y $n = 2^{14}$:

colisiones $\leq$	101	108	119	126	134	145	153
con probabilidad	.009	.043	.244	.476	.742	.946	.989

La teoría señala que estas probabilidades son las mismas que las empleadas en la prueba de póker, ecuación (5); la probabilidad de que ocurran c colisiones es la probabilidad de que $n - c$ urnas estén ocupadas, esto es

$$\frac{m(m-1)\ldots(m-n+c+1)}{m^n} \left\{ {n \atop n-c} \right\}.$$

Aunque m y n son muy grandes, no es difícil calcular estas probabilidades usando el siguiente método:

Algoritmo S.

Dados m y n, este algoritmo determina la distribución del número de colisiones que ocurren cuando n bolas se esparcen en m urnas. Se usa un arreglo auxiliar $A[0]$, $A[1]$, ..., $A[n]$ de números reales para los cálculos necesarios; actualmente $A[j]$ será diferente de cero sólo para $j_0 \leq j \leq j_1$, y $j_1 - j_0$ será a lo más de orden log n, así será posible trabajar con un considerable ahorro de almacenamiento.

S1. Iniciar $A[j] = 0$ para $0 \leq j \leq n$; entonces iniciar $A[j] = 1$ y $j_0 = j_1 = 1$. Después hacer el paso S2 exactamente $n - 1$ veces y continuar con el paso S3.

S2. (Cada vez que se hace este paso equivale a lanzar una bola en una urna; $A[j]$ representa la probabilidad de que exactamente j urnas estén ocupadas).
Hacer $j_1 = j_1 + 1$. Luego para $j = j_1, j_1-1, \ldots, j_0$ (en este orden), hacer $A[j] = (j/m)A[j] + ((1 + 1/m) - (j/m))A[j - 1]$. Si $A[j]$ es muy pequeño como resultado de

estos cálculos, es decir, $A[j] < 10^{-20}$, hacer $A[j] = 0$; y en tal caso, si $j = j_1$ disminuir j_1 en 1, o si $j = j_0$ incrementar j_0 en 1.

S3. En este paso hacemos uso de una tabla auxiliar $(T_1, T_2, \ldots, T_{tmax})$ = (.01, .05, .25, .50, .75, .95, .99, 1.00) conteniendo los porcentajes específicos de interés. Hacer p=0, t = 1, y $j = j_0 - 1$. Hacer la siguiente iteración hasta que t = tmax : incrementar j en 1, y hacer $p = p + A[j]$; después si $p > T_t$, la salida es $n - j - 1$ y $1 - p$ (significa que con probabilidad $1 - p$ hay al menos $n - j - 1$ colisiones) e incrementar repetidamente t por 1 hasta que $p \leq T_t$.

J. Prueba de correlación de series.

Podemos calcular también el siguiente estadístico:

$$C = \frac{n(U_0U_1+U_1U_2+\cdots+U_{n-2}U_{n-1}+U_{n-1}U_0) - (U_0+U_1+\cdots+U_{n-1})^2}{n(U_0^2+U_1^2+\cdots+U_{n-1}^2)-(U_0+U_1+\cdots+U_{n-1})^2} \tag{9}$$

Este es el "coeficiente de correlación de series"; el cual es una medida de la cantidad U_{j+1} que depende de U_j.

El coeficiente de correlación aparece con frecuencia en estadística; si tenemos n cantidades $U_0, U_1, \ldots, U_{n-1}$ y otras n $V_0, V_1, \ldots, V_{n-1}$, el coeficiente de correlación entre ellas se define

$$C = \frac{n\sum(U_jV_j) - (\sum U_j)(\sum V_j)}{\sqrt{(n\sum U_j^2 - (\sum U_j)^2)(n\sum V_j^2 - (\sum V_j)^2)}} \tag{10}$$

Todas las sumatorias en esta fórmula son tomadas sobre el rango $0 \leq j < n$; la ecuación (9) es el caso especial $V_j = U_{(j+1) \bmod n}$. (Nota: El denominador de (10) es cero cuando $U_0 = U_1 = \cdots = U_{n-1}$ o $V_0 = V_1 = \cdots = V_{n-1}$; en nuestra exposición excluimos este

caso).

Un coeficiente de correlación siempre está entre -1 y +1. Cuando es cero o muy pequeño, indica que las cantidades U_j y V_j son independientes, pero cuando el coeficiente de correlación es ±1 indica una dependencia lineal total; en realidad $V_j = \alpha \pm \beta U_j$ para toda j en tal caso, para algunas constantes α y β.

Por lo que se desea tener C en la ecuación (9) cercano a cero. Hasta ahora, como U_0U_1 no son completamente independientes de U_1U_2, el coeficiente de correlación de series no se espera que sea *exactamente* cero. Un "buen" valor de C estará entre $\mu_n - 2\sigma_n$ y $\mu_n + 2\sigma_n$, donde

$$\mu_n = \frac{-1}{n-1}, \qquad \sigma_n = \frac{1}{n-1}\sqrt{\frac{n(n-3)}{n+1}}, \quad n > 2 \qquad (11)$$

Se espera que C este entre estos límites aproximadamente el 95 % de las veces.

Las ecuaciones (11) sólo son mencionadas en este caso, puesto que la distribución exacta de C se desconoce cuando las U's están uniformemente distribuidas. Evidencias empíricas señalan que podemos hacer uso seguro de las fórmulas de la media y desviación estándar que se han deducido de los supuestos de la distribución normal, sin mucho error; estos son los valores correspondientes a la ecuación (11). Se conoce que $\lim_{n\to\infty} \sqrt{n}\sigma_n = 1$; de donde se deducen más resultados generales acerca de las correlaciones de series de secuencias dependientes.

En lugar de simplemente calcular el coeficiente de correlación entre las observaciones (U_0, U_1, ..., U_{n-1}) y sus sucesores inmediatos (U_1, ..., U_{n-1},U_0), podemos también calcularlo entre (U_0, U_1, ..., U_{n-1}) secuencia cambiada cíclicamente (U_q, ..., U_{n-1}, U_0, ..., U_{q-1}); las correlaciones cíclicas serán pequeñas para $0 < q < n$. Un cálculo eficiente de la ecuación (10) para toda q necesitará aproximadamente de n^2 multiplicaciones, pero actualmente es posible calcular todas las correlaciones en sólo $O(n \log n)$ pasos usando la "transformada rápida de Fourier" (Knuth, 1969).

3.3 PRUEBAS TEORICAS

Si bien siempre es posible probar un generador de números aleatorios usando las pruebas empíricas, es mucho mejor contar con "pruebas a priori", es decir, resultados teóricos que nos permitan visualizar anticipadamente qué tan bien saldrán aquellas pruebas. Tales resultados teóricos nos permiten comprender más acerca de los métodos de generación que las empíricas, resultados de "prueba y error". A continuación estudiaremos las secuencias congruenciales lineales com más detalle; si conocemos los resultados de ciertas pruebas antes de generar los números, tenemos una mejor oportunidad para elegir *a*, *m*, y *c* apropiadamente.

El desarrollo de este tipo de teoría es bastante difícil, aunque se han logrado algunos avances. Los resultados obtenidos hasta aquí son generalmente de pruebas estadísticas aplicadas al *periodo completo.* No todas las pruebas estadísticas tienen sentido cuando se aplican a un período completo, por ejemplo, la prueba de frecuencias dará resultados que son demasiado perfectos; pero la prueba de series, la prueba de intervalos, la prueba de permutaciones, la prueba de máximos, etc. pueden analizarse provechosamente en esta forma. Estos estudios detectarán la no aleatoriedad de la secuencia, es decir, el comportamiento inadecuado en muestras muy grandes.

La teoría dada a continuación es bastante clara, pero no elimina la necesidad de probar la no aleatoriedad de una secuencia por los métodos empíricos. Desde luego, parece ser extremadamente difícil probar algo útil acerca de las secuencias cortas. Se conocen pocos resultados teóricos acerca del comportamiento de las secuencias congruenciales lineales con un periodo menor al completo.

Comencemos con una demostración de una simple *ley a priori*, para el caso menos complicado de la prueba de permutaciones. La esencia de nuestro primer teorema es que tenemos $X_{n+1} < X_n$

aproximadamente la mitad de las veces, estipulando que la secuencia es altamente potencial.

Teorema P. Sean a, c, y m que generan una secuencia congruencial lineal con periodo máximo; sea $b=a-1$ y d el máximo común divisor de m y b. La probabilidad de que $X_{n+1} < X_n$ es igual a $\frac{1}{2} + r$, donde

$$r = (2(c \bmod d) - d)/2m \; ; \tag{1}$$

por tanto

$$|r| < d/2m.$$

Demostración. La demostración de este teorema involucra algunas técnicas muy interesantes. Primero definimos

$$s(x) = (ax + c) \bmod m \tag{2}$$

esto es, $X_{n+1} = s(X_n)$, y el teorema se reduce a contar el número de enteros x tales que $0 \le x < m$ y $s(x) < x$ (puesto que cada entero ocurre en alguna parte del periodo). Deseamos mostrar que este número es

$$\tfrac{1}{2}(m + 2(c \bmod d) - d). \tag{3}$$

La función $\lceil (x - s(x))/m \rceil$ es igual a 1 cuando $x > s(x)$, y es 0 en cualquier otro caso; por lo tanto el contador que deseamos obtener puede escribirse simplemente como

$$\sum_{0 \le x < m} \left\lceil \frac{x - s(x)}{m} \right\rceil = \sum_{0 \le x < m} \left\lceil \frac{x}{m} - \left(\frac{ax + c}{m} - \left\lfloor \frac{ax + c}{m} \right\rfloor \right) \right\rceil$$

$$= \sum_{0 \le x <} \left(\left\lfloor \frac{ax + c}{m} \right\rfloor - \left\lfloor \frac{bx + c}{m} \right\rfloor \right). \tag{4}$$

(Recordando que $\lceil -y \rceil = - \lfloor y \rfloor$ y $b = a - 1$) Por resultados obtenidos en la práctica hemos probado que

$$\sum_{0\le j<k} \left\lfloor \frac{hj + c}{k} \right\rfloor = \frac{(h-1)(k-1)}{2} + \frac{g-1}{2} + g\lfloor c/g \rfloor, \qquad (5)$$

$$g = \mathrm{mcd}(h,k)$$

siempre que h y k sean enteros y k>0. Puesto que a es primo relativo de m, esta fórmula produce

$$\sum_{0\le x<m} \left\lfloor \frac{ax + c}{m} \right\rfloor = \frac{(a-1)(m-1)}{2} + c,$$

$$\sum_{0\le x<m} \left\lfloor \frac{bx + c}{m} \right\rfloor = \frac{(b-1)(m-1)}{2} + \frac{d-1}{2} + c - (c \bmod d),$$

y (3) se cumple inmediatamente. ∎

La demostración del teorema P indica que una prueba a *priori* puede, desde luego, realizarse, probando que estamos aptos para tratar satisfactoriamente con sumas involucrando las funciones $\lfloor\ \rfloor$ y $\lceil\ \rceil$. En muchos casos la mayoría de las técnicas poderosas para tratar con funciones piso y techo las remplaza por unas más simétricas:

$$\delta(z) = \lfloor z \rfloor + 1 - \lceil z \rceil = \begin{cases} 1, & \text{si } z \text{ es un entero;} \\ 0, & \text{si } z \text{ no es entero;} \end{cases} \qquad (6)$$

$$((z)) = z - \lfloor z \rfloor - \tfrac{1}{2} + \tfrac{1}{2}\delta(z) = z - \lceil z \rceil + \tfrac{1}{2} - \tfrac{1}{2}\delta(z). \qquad (7)$$

La segunda función es una función "diente de sierra" familiar en el estudio de series de Fourier; su gráfica se muestra en la figura 3.3. La razón para trabajar con $((z))$ en lugar de $\lfloor z \rfloor$ es que $((z))$ posee varias propiedades muy útiles:

$$((-z)) = -((z)); \qquad (8)$$

$$((z + n)) = ((z)), \quad n \text{ entero}; \qquad (9)$$

$$((nz)) = ((z)) + \left(\left(z + \frac{1}{n}\right)\right) + \cdots + \left(\left(z + \frac{n-1}{n}\right)\right), \qquad (10)$$

n entero ≥ 1.

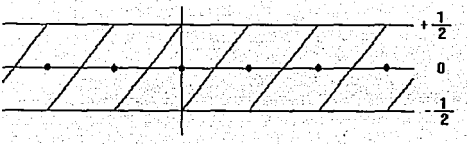


Figura 3.3
Función diente de sierra $\{(x)\}$.

Para obtener algunos trabajos prácticos con estas funciones, probemos el teorema P otra vez. Con ayuda de las ecuaciones (7), (8), (9), podemos mostrar que

$$\left\lceil \frac{x - s(x)}{m} \right\rceil = \frac{x - s(x)}{m} - \left(\left(\frac{x - s(x)}{m} \right) \right) + \frac{1}{2} - \frac{1}{2}\delta\left(\frac{x - s(x)}{m} \right)$$

$$= \frac{x - s(x)}{m} - \left(\left(\frac{x - (ax + c)}{m} \right) \right) + \frac{1}{2}$$

$$= \frac{x - s(x)}{m} - \left(\left(\frac{bx + c}{m} \right) \right) + \frac{1}{2} \qquad (11)$$

puesto que $(x - s(x))/m$ nunca es entero. Ahora

$$\sum_{0 \le x < m} \frac{x - s(x)}{m} = 0$$

dado que x y $s(x)$ toman exactamente un valor de $\{0, 1, \ldots, m - 1\}$; por lo tanto (11) da

$$\sum_{0\le x<m} \left\lceil \frac{x - s(x)}{m} \right\rceil = \sum_{0\le x<m} \left(\left(\frac{bx + c}{m} \right) \right) + \frac{m}{2}. \qquad (12)$$

Sea $b = b_0 d$, $m = m_0 d$, donde b_0 y m_0 son primos relativos. Sabemos que $(b_0 x) \bmod m_0$ toma los valores $\{0, 1, \ldots, m_0 - 1\}$ en el orden en que x varia desde 0 hasta $m - 1$. Por (9) y (10) y el hecho de que

$$\left(\left(\frac{b(x + m_0) + c}{m} \right) \right) = \left(\left(\frac{bx + c}{m} \right) \right)$$

tenemos

$$\sum_{0\le x<m} \left(\left(\frac{bx + c}{m} \right) \right) = d \sum_{0\le x<m} \left(\left(\frac{bx + c}{m} \right) \right)$$

$$= d \sum_{0\le x<m} \left(\left(\frac{c}{m} + \frac{b_0 x}{m} \right) \right) = d\left(\left(\frac{c}{d} \right) \right). \qquad (13)$$

El teorema P resulta inmediatamente de (12) y (13).

Una consecuencia del teorema P es que prácticamente cualquier elección de a y c dará una probabilidad rasonable de que $X_{n+1} < X_n$, al menos sobre el periodo completo, excepto aquellos que tienen d grande. Un valor grande de d corresponde a una potencia baja, y ya sabemos que los generadores de baja potencia son indeseables.

El siguiente teorema nos da una condición más estricta para la elección de *a* y *c*; consideraremos la *prueba de correlación serial* aplicada sobre el periodo completo. La cantidad C definida en la sección 3.2, ecuación (9), es:

$$C = \left(m \sum_{0\le x<m} x s(x) - \left(\sum_{0\le x<m} x \right)^2 \right) \Big/ \left(m \sum_{0\le x<m} x^2 - \left(\sum_{0\le x<m} x \right)^2 \right). \qquad (14)$$

Sea x' el elemento tal que $s(x') = 0$. Tenemos

$$s(x) = m\left(\left(\frac{ax + c}{m}\right)\right) + \frac{m}{2}, \qquad \text{si } x \neq x'. \tag{15}$$

Las fórmulas que aproximadamente deducimos pueden expresarse más fácilmente en términos de la función

$$\sigma(h,k,c) = 12 \sum_{0 \le j < k} \left(\left(\frac{j}{k}\right)\right)\left(\left(\frac{hj + c}{k}\right)\right), \tag{16}$$

función muy importante que surge de varios problemas matemáticos. Esta es denominada una *suma generalizada* de Dedekind, puesto que Richard Dedekind introdujo la función $\sigma(h,k,0)$ en 1876 cuando comentó uno de los manuscritos incompletos de Riemann.

Usando las fórmulas bien conocidas

$$\sum_{0 \le x < m} x = \frac{m(m-1)}{2} \qquad \text{y} \qquad \sum_{0 \le x < m} x^2 = \frac{m(m-1)(2m-1)}{6},$$

esta es una forma directa de transformar la ecuación (14) a

$$C = \frac{m\sigma(a,m,c) - 3 + 6(m - x' - c)}{m^2 - 1}. \tag{17}$$

Puesto que m es comunmente muy grande, podemos descartar términos del orden $1/m$, y obtener la transformación

$$C \approx \sigma(a,m,c)/m, \tag{18}$$

con un error menor que $6/m$ en valor absoluto.

La prueba de correlación de series ahora se reduce determinando el valor de la suma de Dedekind. La evaluación $\sigma(a,m,c)$ directamente de la definición (16) es dificilmente más sencillo que evaluar el coeficiente de correlación directamente, pero afortunadamente hay métodos más simples para calcular las sumas de Dedekind rápidamente.

Lema B. (Ley de reciprocidad de las sumas de Dedekind). Sean h, k, c, enteros. Si $0 \le c < k$, $0 < h \le k$, y si h es primo relativo de k, entonces

$$\sigma(h,k,c) + \sigma(k,h,c) = \frac{h}{k} + \frac{k}{h} + \frac{1}{hk} + \frac{6c^2}{hk} - 6\left\lfloor \frac{c}{h} \right\rfloor - 3e(h,c), \qquad (19)$$

donde

$$e(h,c) = \begin{cases} 1, & \text{si } c = 0 \quad \text{o} \quad c \bmod h \neq 0; \\ 0, & \text{si } c > 0 \quad \text{y} \quad c \bmod h = 0. \end{cases} \qquad (20)$$

Demostración. Este lema lo demostraremos sólo para el caso c=0. Esta demostración, está basada en las raices complejas de la unidad, esencialmente la hizo L. Carlitz. Actualmente hay una demostración más simple que usa sólo manipulaciones de sumas elementales pero el método siguiente revela más de las herramientas matemáticas que están disponibles para problemas de este tipo y son por lo tanto mucho más instructivas.

Sean $f(x)$ y $g(x)$ los polinomios definidos como sigue:

$$\begin{aligned} f(x) &= 1 + x + \cdots + x^{k-1} = (x^k - 1)/(x - 1) \\ g(x) &= x + 2x^2 + \cdots + (k-1)x^{k-1} = xf'(x) \\ &= kx^k/(x-1) - x(x^k - 1)/(x-1)^2. \end{aligned} \qquad (21)$$

Si ω es la k-ésima raíz compleja de la unidad $e^{2\pi i/k}$, por la ecuación

$$\sum_{k \bmod m = r} a_k z^k = \frac{1}{m} \sum_{1 \le j < m} \omega^{-jr} G(\omega^j z), \qquad 0 \le r < m$$

donde

$$\omega = e^{2\pi i/m}$$

tenemos

$$\frac{1}{k} \sum_{0 \le j < k} \omega^{-jr} g(\omega^j x) = r x^r, \qquad \text{si } 0 \le r < k. \qquad (22)$$

Sea $x = 1$; entonces $g(\omega^j x) = k/(\omega^j - 1)$ si $j \neq 0$, en otro caso es igual a $k(k-1)/2$, por tanto

$$r \bmod k = \sum_{0<j<k} \frac{\omega^{-jr}}{\omega^j - 1} + \frac{1}{2}(k-1), \qquad \text{si } r \text{ es un entero.}$$

(La ecuación (22) muestra que el lado derecho es igual a r cuando $0 \le r < k$, y este se intercambia cuando los múltiplos de k son agregados a r). Por lo tanto

$$((\frac{r}{k})) = \frac{1}{k} \sum_{0<j<k} \frac{\omega^{-jr}}{\omega^j - 1} - \frac{1}{2k} + \frac{1}{2}\sigma(\frac{r}{k}). \qquad (23)$$

Esta importante fórmula, la cual es válida cuando r es un entero, nos permite reducir muchos cálculos involucrando $((r/k))$ para sumas que involucran la raíz k-ésima de la unidad, y proporciona un rango completo de técnicas en la descripción. En particular, obtuvimos la siguiente formula:

$$\sigma(h,k,0) + \frac{3(k-1)}{k^2} = \frac{12}{k^2} \sum_{0<r<k} \sum_{0<l<k} \sum_{0<j<k} \frac{\omega^{-lr}}{\omega^l - 1} \frac{\omega^{-jhr}}{\omega^j - 1}. \qquad (24)$$

El lado derecho de esta fórmula puede simplificarse realizando la suma sobre r; tenemos $\sum_{0 \le r < k} \omega^{rs} = f(\omega^s) = 0$ si $s \bmod k \neq 0$. La ecuación (24) ahora se reduce a

$$\sigma(h,k,0) + \frac{3(k-1)}{k} = \frac{12}{k} \sum_{0<j<k} \frac{1}{(\omega^{-jh} - 1)(\omega^j - 1)} \qquad (25)$$

Una fórmula similar se obtiene para $\sigma(k,h,0)$, con $\zeta = e^{2\pi i/h}$ remplazando ω.

No es obvio lo que podemos hacer con la suma en (25), pero hay una forma elegante de proceder, basados en el hecho de que cada término de la suma es una función de ω^j, donde $0 < j < k$; por lo tanto la suma toma esencialmente las k-ésimas raices de la unidad mayores que 1. Sin embargo $x_1, x_2, \ldots, x_n$ son números complejos distintos, tenemos la identidad

$$\sum_{1\le j\le n} \frac{1}{(x_j - x_1)\cdots(x_j - x_{j-1})(x - x_j)(x_j - x_{j+1})\cdots(x_j - x_n)}$$

$$= \frac{1}{(x - x_1)\cdots(x - x_n)}, \qquad (26)$$

la cual sigue el método usual de expansión del lado derecho de fracciones parciales. Por otra parte, si $q(x) = (x - y_1)(x - y_2)\cdots(x - y_m)$, tenemos

$$q'(y_j) = (y_j - y_1)\ldots(y_j - y_{j-1})(y_j - y_{j+1})\ldots(y_j - y_m); \qquad (27)$$

esta identidad frecuentemente puede usarse para simplificar expresiones como aquellas del lado izquierdo de (26). Cuando h y k son primos relativos, los números ω, ω^2, ..., ω^{k-1}, ζ, ζ^2, ..., ζ^{h-1} son todos distintos; por lo tanto podemos considerar la fórmula (27) en el caso especial del polinomio $(x - \omega)\ldots(x - \omega^{k-1})(x - \zeta)\ldots(x - \zeta^{h-1}) = (x^k - 1)(x^h - 1)/(x - 1)^2$, obteniendo la siguiente identidad en x:

$$\frac{1}{h}\sum_{0<j<h} \frac{\zeta^j(\zeta^j - 1)^2}{(\zeta^{jk} - 1)(x - \zeta^j)} + \frac{1}{k}\sum_{0<j<k} \frac{\omega^j(\omega^j - 1)^2}{(\omega^{jh} - 1)(x - \omega^j)} = \frac{(x - 1)^2}{(x^h - 1)(x^k - 1)}. \qquad (28)$$

Esta identidad tiene muchas consecuencias interesantes, y lleva a numerosas formulas recíprocas para las sumas del tipo dado en la ecuación (25). Por ejemplo, si diferenciamos (28) dos veces con respecto a x y asignamos $x \to 1$, encontramos que

$$\frac{2}{h}\sum_{0<j<h} \frac{\zeta^j(\zeta^j - 1)^2}{(\zeta^{jk} - 1)(1 - \zeta^j)^3} + \frac{1}{k}\sum_{0<j<k} \frac{\omega^j(\omega^j - 1)^2}{(\omega^{jh} - 1)(1 - \omega^j)^3}$$

$$= \frac{1}{6}\left(\frac{h}{k} + \frac{k}{h} + \frac{1}{hk}\right) + \frac{1}{2} - \frac{1}{2h} - \frac{1}{2k}.$$

Remplazando j por $h - j$ y por $k - h$ en esta suma y usando (25) obtenemos

$$\frac{1}{6}\left(\sigma(k,h,0) + \frac{3(h-1)}{h}\right) + \frac{1}{6}\left(\sigma(h,k,0) + \frac{3(k-1)}{k}\right)$$

$$= \frac{1}{6}\left(\frac{h}{k} + \frac{k}{h} + \frac{1}{hk}\right) + \frac{1}{2} - \frac{1}{2h} - \frac{1}{2k},$$

lo cual es equivalente al resultado deseado. ■

El lema B nos da una función explícita $f(h,k,c)$ tal que

$$\sigma(h,k,c) = f(h,k,c) - \sigma(k,h,c) \tag{29}$$

cuando $0 < h \le k$, $0 \le c < k$, y h es primo relativo de k. De la definición (16) es claro que

$$\sigma(k,h,c) = \sigma(k \bmod h,\ h,\ c \bmod h). \tag{30}$$

Por tanto podemos usar (29) iterativamente para evaluar $\sigma(h,k,c)$, usando un proceso que reduce los parámetros como en el algoritmo de Euclides.

Se dan más simplificaciones cuando examinamos este procedimiento iterativo más exhaustivamente. Asignemos $m_1 = k$, $m_2 = h$, $c_1 = c$, y formamos el siguiente tableau:

$$\begin{aligned} m_1 &= a_1 m_2 + m_3 & \qquad c_1 &= b_1 m_2 + c_2 \\ m_2 &= a_2 m_3 + m_4 & \qquad c_2 &= b_2 m_3 + c_3 \\ m_3 &= a_3 m_4 + m_5 & \qquad c_3 &= b_3 m_4 + c_4 \\ m_4 &= a_4 m_5 & \qquad c_4 &= b_3 m_4 + c_5 \end{aligned} \tag{31}$$

Aquí

$$a_j = \lfloor m_j / m_{j+1} \rfloor, \qquad b_j = \lfloor c_j / m_{j+1} \rfloor,$$

$$m_{j+2} = m_j \bmod m_{j+1}, \qquad c_{j+1} = c_j \bmod m_{j+1}, \tag{32}$$

y resulta que

$$0 \le m_{j+1} < m_j, \qquad 0 \le c_j < m_j. \tag{33}$$

Hemos asumido por conveniencia que el algoritmo de Euclides termina en (31) después de cuatro iteraciones; esta suposición revelará el patrón que se mantiene en el caso general. Puesto que h y k eran primos relativos al iniciar, debemos tener $m_5 = 1$ y $c_5 = 0$ en (31).

En adelante suponemos que $c_3 \neq 0$ pero $c_4 = 0$, para obtener un ligero pronóstico del efecto que este tiene en la recurrencia. Las ecuaciones (29) y (30) dan

$$\begin{aligned}\sigma(h,k,c) &= \sigma(m_2,m_1,c_1)\\ &= f(m_2,m_1,c_1) - \sigma(m_3,m_2,c_2)\\ &= \cdots\\ &= f(m_2,m_1,c_1) - f(m_3,m_2,c_2) + f(m_4,m_3,c_3) - f(m_5,m_4,c_4).\end{aligned}$$

La primera parte "$h/k + k/h$" de la formula para $f(h,k,c)$ en (19) contribuye

$$\frac{m_2}{m_1} + \frac{m_1}{m_2} - \frac{m_3}{m_2} - \frac{m_2}{m_3} + \frac{m_4}{m_3} + \frac{m_3}{m_4} - \frac{m_5}{m_4} - \frac{m_4}{m_5}$$

al total, y esto se reduce a

$$\frac{h}{k} + \left(a_1 + \frac{m_3}{m_2}\right) - \frac{m_3}{m_2} - \left(a_2 + \frac{m_4}{m_3}\right) + \frac{m_4}{m_3} + \left(a_3 + \frac{m_5}{m_4}\right) - \frac{m_5}{m_4} - a_4$$

$$= h/k + a_1 - a_2 + a_3 - a_4.$$

La siguiente parte "$1/hk$" de (19) también deja una contribución sencilla; de acuerdo a la ecuación:

$$/x_1,\ldots,x_n/ = \frac{1}{q_0q_1} - \frac{1}{q_1q_2} + \frac{1}{q_2q_3} - \ldots + \frac{1}{q_{n-1}q_n}$$

donde $q_k = Q_k(x_1,\ldots,x_k)$

y

$$Q_n(x_1,x_2,\ldots,x_n) = \begin{cases} 1 & \text{si } n=0; \\ x_1 & \text{si } n=1; \\ x_1Q_{n-1}(x_2,\ldots,x_n)+Q_{n-2}(x_3,\ldots,x_n), & \text{si } n>1. \end{cases}$$

tenemos

$$1/m_1m_2 - 1/m_2m_3 + 1/m_3m_4 - 1/m_4m_5 = h'/k - 1, \qquad (34)$$

donde h' es el único entero que satisface

$$h'h \equiv 1 \text{ (modulo k)}, \quad 0 < h' \le k. \qquad (35)$$

agregando esto a lo anterior y recordando el supuesto de que $c_4=0$ (así que $e(m_4,c_3) = 0$), se encuentra que

$$\sigma(h,k,c) = \frac{h+h'}{k} + (a_1 - a_2 + a_3 - a_4) - 6(b_1 - b_2 + b_3 - b_4)$$
$$+ 6\left(\frac{c_1^2}{m_1m_2} - \frac{c_2^2}{m_2m_3} + \frac{c_3^2}{m_3m_4} - \frac{c_4^2}{m_4m_5}\right) + 2,$$

en términos del tableau supuesto (31). Los resultados que se mantienen en general:

Teorema D. Sean h,k,c enteros con $0 < h \le k$, $0 \le c < k$, y h primo relativo de k. Forman el "tableau euclideano" como se definió en (32), y suponiendo que el proceso se detiene después de t pasos con $m_{t+1} = 1$. Sea s el valor más pequeño tal que $c_s = 0$, y sea h' definido por (35). Entonces

$$\sigma(h,k,c) = \frac{h+h'}{k} + \sum_{1\le j\le t} (-1)^{j+1}\left(a_j - 6b_j + 6\,\frac{c_j^2}{m_jm_{j+1}}\right)$$

$$+ 3((-1)^a + \delta_{s1}) - 2 + (-1)^t. \qquad \blacksquare$$

Ahora que hemos analizado las sumas generalizadas de Dedekind, aplicaremos nuestros conocimientos en la determinación de los coeficientes de correlación de series.

Ejemplo 1. Encontrar la *correlación de series* cuando $m = 2^{35}$, $a = 2^{34} + 1$, $c = 1$.

Solución. Tenemos

$$C = (2^{35}\sigma(2^{34}+1,\ 2^{35},\ 1) - 3 + 6(2^{35} - (2^{34} - 1) - 1))/(2^{70} - 1)$$

por la ecuación (17). Para evaluar $\sigma(2^{34} + 1,\ 2^{35},\ 1)$, podemos formar el tableau

$$\begin{array}{llll}
m_1 = 2^{35} & & c_1 = 1 & \\
m_2 = 2^{34} + 1 & a_1 = 1 & c_2 = 1 & b_1 = 0 \\
m_3 = 2^{34} - 1 & a_2 = 1 & c_3 = 1 & b_2 = 0 \\
m_4 = 2 & a_3 = 2^{33} - 1 & c_4 = 1 & b_3 = 0 \\
m_5 = 1 & a_4 = 2 & c_5 = 0 & b_4 = 1
\end{array}$$

Puesto que $h' = 2^{34} + 1$, el valor de acuerdo al teorema D se convierte en $2^{33} - 3 + 2^{-32}$. Esto es

$$C = (2^{68} + 5)/(2^{70} - 1) = \tfrac{1}{4} + \varepsilon, \qquad |\varepsilon| < 2^{-67}. \qquad (36)$$

Tal correlación es mucho, mucho más alta para la aleatoriedad. De hecho, este generador tiene muy bajo potencial, y ya lo hemos rechazado como no aleatorio.

Ejemplo 2. Encontrar la *correlación de series aproximada* cuando $m = 10^{10}$, $a = 10001$, $c = 2113248653$.

Solución. Tenemos $C \approx \sigma(a,m,c)/m$, y el cálculo procede como sigue:

$$\begin{array}{llllll}
m_1 = 10000000000 & & & c_1 = 2113248653 & & \\
m_2 = 10001 & a_1 = 999900 & & c_2 = 7350 & b_1 = 211303 & \\
m_3 = 100 & a_2 = 100 & & c_3 = 50 & b_2 = 73 & \\
m_4 = 1 & a_3 = 100 & & c_4 = 0 & b_3 = 50 &
\end{array}$$

$$\sigma(m_2,m_1,c_1) = -31.6926653544;$$

$$C \approx -3\cdot 10^{-9} \qquad (37)$$

Desde luego, este valor de C es muy respetable. Pero el generador tiene una potencia de sólo 3, así que este no es realmente una buena fuente de números aleatorios a pesar de que tiene baja correlación de series. Es necesario tener una baja correlación de series, pero no es suficiente.

Ejemplo 3. Estimar la *correlación de series* para cualquier a, m, y c.

Solución. Si consideramos sólo una aplicación de (29), tenemos

$$\sigma(a,m,c) \approx \frac{m}{a} + 6\frac{c^2}{am} - 6\frac{c}{a} - \sigma(m,a,c).$$

Ahora $|\sigma(m,a,c)| < a$, y por tanto

$$C \approx \frac{\sigma(a,m,c)}{m} \approx \frac{1}{a}\left(1 - 6\frac{c}{m} + \left(\frac{c}{m}\right)^2\right). \qquad (38)$$

El error en esta aproximación es menor que $(a + 6)/m$ en valor absoluto.

La estimación (38) fue el primer resultado teórico conocido acerca de la aleatoriedad de generadores congruenciales. R. R. Coveyou la obtuvo promediando todos los números *reales* x entre 0 y m en lugar de considerar sólo los valores enteros; después Martín Greenberger da un resultado riguroso que incluye la estimación del término del error.

Así comenzó uno de los capítulos más tristes en la historia de la ciencia de las computadoras. Aunque la aproximación anterior es bastante correcta, en la práctica esta ha sido gravemente distorsionada; la gente abandonó los generadores perfectamente buenos, éstos fueron usados y remplazados por generadores fatales que se han observado buenos desde el punto de vista de la ecuación (38). Por más de una década, la mayoría de generadores de números

aleatorios de uso diario eran seriamente ineficientes, debido únicamente a un avance teórico.

Si aprendemos de errores pasados, tenemos que ver más cuidadosamente cómo la ecuación (38) ha sido mal empleada. En primer lugar la gente supone inexpertamente que una correlación de series pequeña sobre el período completo debería ser una buena garantía de aleatoriedad; pero de hecho esto aún no asegura una correlación de series pequeña para 1000 elementos consecutivos de una secuencia.

En segundo, la ecuación (38) y su término de error asegurarán relativamente un valor pequeño de C sólo cuando $a \approx \sqrt{m}$, por tanto la gente sugirió elegir multiplicadores cercanos a $\sqrt{m}$. De echo, veremos que casi todos los multiplicadores dan un valor de C que es esencialmente menor que $1/\sqrt{m}$, por lo tanto (38) no es una muy buena aproximación al verdadero comportamiento. Minimizando un simple límite superior para C no lo minimiza.

En tercer lugar, la gente observó que (38) da su mejor estimación cuando $c/m \approx \frac{1}{2} \pm \frac{1}{6}\sqrt{3}$, por lo tanto estos valores son las raices de $1 - 6x + 6x^2 = 0$. En ausencia de cualquier otro criterio para la selección de c, debemos usar éste. La última declaración no es correcta, pero engaña al mejor, puesto que la experiencia ha mostrado que el valor de c tiene difícilmente alguna influencia del verdadero valor de la correlación de series cuando a es un buen multiplicador; la elección $c/m \approx \frac{1}{2} \pm \frac{1}{6}\sqrt{3}$ reduce C sustancialmente sólo en casos como el del ejemplo 2 anterior. Y en tales casos nos engañamos a nosotros mismos, puesto que el mal multiplicador revelará sus deficiencias en otras formas.

Claramente necesitamos un mejor estimador que (38); y tal estimador ahora está disponible gracias al teorema D, el cual tiene sus bases principalmente en el trabajo de U. Dieter. El teorema D implica que $\sigma(a,m,c)$ será más pequeña si las cantidades parciales de a/m son pequeñas. Desde luego, analizando las sumas generalizadas de Dedekind aún más exhaustivamente P, es posible

obtener un estimador bastante aproximado.

Teorema K. Bajo los supuestos del teorema D, siempre tenemos

$$-\frac{1}{2}\sum_{1\le j\le t\ j\ \text{impar}} a_j - \sum_{1\le j\le t\ j\ \text{par}} a_j + \frac{1}{2} \le \sigma(h,k,c) \le \sum_{1\le j\le t\ j\ \text{impar}} a_j + \frac{1}{2}\sum_{1\le j\le t\ j\ \text{par}} a_j - \frac{1}{2} \qquad (39)$$

Ejemplo 4. Estimar la correlación de series para $a = 3141592621$, $m = 2^{35}$, c impar.

Solución. Las cantidades parciales de a/m son 10,1,14,1,7,1,1,1,3,3,3,5,2,1,8,7,1,4,1,2,4,2; por lo tanto por el teorema K

$$-45 \le \sigma(a,m,c) \le 68,$$

y se garantiza que la correlación de series es extremadamente pequeña para toda c.

Observemos que este límite es considerablemente mejor que el que podríamos obtener de (38), puesto que el error en (38) es del orden a/m; nuestro multiplicador "aleatorio" ha resultado ser mucho mejor que uno elegido específicamente observando bien las bases de (38). De hecho, es posible mostrar que el valor promedio de $\sum_{1\le j\le t} a_j$, tomando todos los multiplicadores a que son primos relativos de m, es

$$\frac{6}{\pi^2}(\ln m)^2 + O((\log m)(\log\log m)^4)$$

Por tanto la probabilidad de que un multiplicador aleatorio tenga $\sum_{1\le j\le t} a_j$ grande, es decir más grande que $(\log m)^{2+\varepsilon}$ para algún $\varepsilon > 0$ establecido, aproximándose a cero conforme $m \to \infty$. Esto comprueba la evidencia empírica de que casi todas las secuencias congruenciales lineales poseen correlaciones de series extremadamente pequeñas en el periodo completo.

3.4 LA PRUEBA ESPECTRAL

En esta sección estudiaremos una forma especialmente importante para verificar la calidad de los generadores de números aleatorios congruenciales lineales. Por mucho esta es la prueba más poderosa que se conoce, y merece especial atención.

La prueba espectral comprende aspectos de las pruebas empíricas y de las teóricas: es como las pruebas teóricas debido a su trato con propiedades del periodo completo de la secuencia, y es como las empíricas debido a que requiere un programa de computadora para determinar los resultados.

3.4.1 Ideas fundamentales de la prueba.

El criterio más importante de aleatoriedad cuenta con propiedades de la distribución conjunta de t elementos consecutivos de la secuencia, y la prueba espectral trata directamente con esta distribución. Si tenemos una secuencia $\langle U_n \rangle$ de periodo m, la idea es analizar el conjunto de m puntos

$$\{ (U_2, U_{n+1}, \ldots, U_{n+t-1}) \} \qquad (1)$$

en el espacio t dimensional.

Por simplificar supondremos que tenemos una secuencia congruencial lineal (X_0, a, c, m) de longitud de periodo máximo m (así que $c \neq 0$), o m es primo relativo y $c = 0$ y la longitud del periodo es $m - 1$. En el siguiente caso agregaremos el punto (0, 0, ..., 0) al conjunto (1), de modo que hay m puntos en total; este punto extra tiene un efecto insignificnte cuando m es grande, y hace mucho más sencilla la teoría. Bajo estos supuestos, (1) puede reescribirse como

$$\{ \frac{1}{m} (x, s(x), s(s(x)), \ldots, s^{t-1}(x)) \mid 0 \leq x < m \}, \qquad (2)$$

donde

$$s(x) = (ax + c) \bmod m \qquad (3)$$

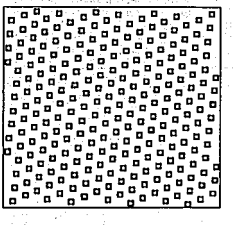
es el "sucesor" de x. Notemos que se consideró sólo el conjunto de todos los puntos en dimensiones t, no el orden en el que estos puntos son generados. Pero el orden de generación se refleja en la dependencia entre componentes de los vectores; y la prueba espectral estudia tal dependencia para varias dimensiones t tratando con la totalidad de puntos (2).

Por ejemplo, la figura 3.4 muestra un caso pequeño típico en 2 y 3 dimensiones, para el generador con

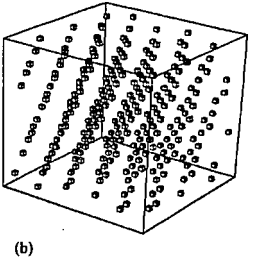
$$s(x) = (137x + 187) \bmod 256. \qquad (4)$$

De hecho un generador con longitud de periodo 256 difícilmente será aleatorio, pero 256 es suficientemente pequeño para que dibujemos el diagrama y logremos un acuerdo antes de recurrir a los m's más grandes.

Tal vez el aspecto más importante del patrón de los cuadros de la figura 3.4 es que podemos cubrir todos por un número pequeño de líneas paralelas; desde luego, hay varias familias diferentes de líneas paralelas que darán en todos los puntos. Por ejemplo,



(a)



(b)

Figura 3.4

a) Rejilla bidimensional formada por todos los pares de puntos sucesivos $(Xn, Xn+1)$, cuando $Xn+1=(137Xn+187) \bmod 256$. b) Rejilla tridimensional de triadas $(Xn, Xn+1, Xn+2)$.

aproximadamente 20 líneas verticales serán suficientes, lo que deja un conjunto de 21 líneas que tienen un ángulo de inclinación de aproximadamente 30 grados.

Si se considera el mismo generador en tres dimensiones, obtendremos 256 puntos en un cubo, obtenido al agregar un "octavo" componente $s(s(x))$ para cada uno de los 256 puntos $(x,s(x))$ en el plano de la figura 3.4 (a), como se muestra en la figura 3.4 (b), Imaginemos que esta estructura de cristal tridimensional ha sido construida en un modelo físico, un cubo que podemos modificar en nuestras manos; así como rotarlo, notaremos varias familias de paralelos planos que abarcan todos los puntos. En las palabras de Wallace Givens: "los números aleatorios permanecen principalmente en los planos."

A primera vista podemos pensar que tal comportamiento sistemático es no aleatorio así que hace a los generadores congruenciales bastante inútiles; pero reflexionando más cuidadosamente, recordemos que m es bastante grande en la práctica, esto proporciona un mejor visión. La estructura regular en la figura 3.4 es esencialmente el "grano" (si el comportamento de las secuencias las comparamos con un sembradío) que podemos examinar en nuestros números aleatorios bajo un microscopio de alto poder. Si tomamos verdaderos números aleatorios entre 0 y 1, y los redondeamos o truncamos para tener una precisión finita de manera que cada uno de los múltiplos enteros de $1/\nu$ para cualquier número dado ν, entonces los puntos t-dimensionales (1) que obtuvimos tendrán un carácter extremadamente regular cuando sean vistos a través de un microscopio.

Sea $1/\nu^2$ la distancia máxima entre líneas, tomando aproximadamente todas las familias de líneas paralelas que cubren los puntos $\{(x/m,s(x)/m)\}$ en dos dimensiones. Llamaremos ν_2 la *exactitud* bi-dimensional del generador de números aleatorios, puesto que las parejas de números sucesivos tienen una estructura fina que es esencialmente buena para una parte de ν_2. Similarmente, sea $1/\nu_3$ la distancia máxima entre planos, tomada sobre todas las familias de planos paralelos que cubren todos los puntos $\{(x/m,s(x)/m,s(s(x))/m)\}$; llamaremos ν_3 a la exactitud en

tres dimensiones. La exactitud t-dimensional ν_t es el recíproco de la distancia máxima entre hiperplanos, tomado sobre todas las familias de hiperplanos paralelos $(t-1)$-dimensionales que cubren todos los puntos $\{(x/m, s(x)/m, \ldots, s^{t-1}(x)/m)\}$.

La diferencia esencial entre secuencias periódicas y secuencias verdaderamente aleatorias que han sido truncadas a multiplos de $1/\nu$ es que la "exactitud" de las secuencias verdaderamente aleatorias es la misma en todas las dimensiones, mientras que la "exactitud" de secuencias periodicas decrece conforme crece t. Desde luego, puesto que hay sólo m puntos en el cubo t-dimensional cuando m es la longitud del periodo, no podemos lograr una exactitud t-dimensional de mayor aproximación que $m^{1/t}$.

Cuando se considera la independencia de t valores consecutivos, los números aleatorios generados por computadora se comportarán esencialmente como si vieramos números verdaderamente aleatorios y truncados a lg ν_t bits, donde ν_t decrece cuando crece t. En la práctica, tal variación de la exactitud es usualmente todo lo que necesitamos. No insistimos en que la exactitud 10-dimensional sea 2^{35}, en el sentido de que todo $(2^{35})^{10}$ posible 10-tuples $(U_n, U_{n+1}, \ldots, U_{n+9})$ debería ser igualmente semejante en una máquina de 35-bits; para tales valores grandes de t deseamos sólo un poco de los bits llevados de $(U_n, U_{n+1}, \ldots, U_{n+t-1})$ a una conducta como si fueran independientemente aleatorios.

De otra manera cuando una aplicación demanda alta resolución de la secuencia de números aleatorios, las secuencias congruenciales lineales simples necesariamente serán inadecuadas; un generador con periodo grande deberá ser usado en su lugar, aún cuando sólo una pequeña fracción del periodo actualmente sea generada. Ajustando el periodo esencialmenteajustamos la exactitud en dimensiones más altas, es decir, doblaremos el número efectivo de bits de precisión.

La prueba espectral se basa en los valores de ν_t para t pequeños, es decir $2 \le t \le 6$. Dimensiones 2, 3 y 4 parecen ser adecuadas para detectar importantes deficiencias en una secuencia, pero puesto que estamos considerando el periodo completo parece mejor ser algo cautelosos e ir en otra dimension o dos; en la otra

forma los valores de ν_t para $t \geq 10$ parece ser de ninguna significancia práctica. (Esto es conveniente, porque los cálculos son más difíciles cuando $t \geq 10$).

Notemos que hay una relación vaga entre la prueba espectral y la prueba de series; por ejemplo, un caso especial de la prueba de series, tomado sobre el periodo completo, cuenta los números de cuadros en cada uno de los 64 subcuadros de la figura 3.4. La diferencia principal es que la prueba espectral rota los puntos así como descubre la orientación menos favorable.

Al principio puede parecer que deberíamos aplicar la prueba espectral sólo para un valor apropiado de t; si un generador pasa la prueba en tres dimensiones, parece aceptable que también pase la prueba en 2-D, por tanto podríamos omitirla después. La falacia en este razonamiento ocurre debido a que aplicamos mas condiciones insignificantes en bajas dimensiones. Una situación similar ocurre con la prueba de series: Considerando un generador que (apropiadamente) tiene casi el mismo número de puntos en cada subcubo del cubo unitario, cuando el cubo ha sido dividido en 64 subcubos de tamaño $\frac{1}{4} \times \frac{1}{4} \times \frac{1}{4}$; este mismo generador debería producir completamente subcuadrados vacíos de la unidad cuadrada, cuando la unidad cuadrada se ha dividido en 64 subcuadros de tamaño $\frac{1}{8} \times \frac{1}{8}$. Puesto que incrementamos nuestro esperanza en dimensiones bajas, se requiere una prueba por separado para cada dimensión.

No siempre es cierto que $\nu_t \leq m^{1/t}$, aunque este límite superior es válido para los puntos de una rejilla rectangular. Por ejemplo, este da que $\nu_2 = \sqrt{274} > \sqrt{256}$ en la figura 3.4, debido a una estructura aproximadamente hexagonal da los m puntos casi juntos que sería posible en un arreglo estrictamente rectangular.

Para desarrollar un algoritmo que calcule ν_t eficientemente, debemos observar más profundamente la teoría matemática asociada. De otra manera, veremos que las matemáticas bajo la teoría espectral requiere sólo algunas manipulaciones elementales de vectores.

Algunos autores han sugerido usar en valor mínimo de N_t de

líneas paralelas e hiperplanos como criterio, en lugar de la distancia máxima $1/\nu_t$ entre ellos. Sin embargo, este número no parece ser tan importante como el concepto de exactitud definido anteriormente, debido a que está basado en la aproximación de la pendiente de las líneas o hiperplanos que marcan los ejes de coordenadas del cubo. Por ejemplo, las aproximadamente 20 líneas verticales que cubren todos los puntos de la figura 3.4 están actualmente separadas $1/\sqrt{238}$ unidades, esto podría falsamente implicar una exactitud de una fracción en $\sqrt{238}$, o tal vez aún en una parte de 20. La verdadera exactitud de sólo una fracción de $\sqrt{274}$ se realiza sólo para la familia de 21 líneas con una pendiente de 7/15; otra familia de 24 lineas, con una pendiente de -11/13, también tiene una distancia interlíneas más grande que la familia de 20 líneas, puesto que $1/\sqrt{290} > 1/\sqrt{328}$. la forma precisa en la cual las familias de las líneas actúan en los límites de la unidad hipercúbica no parece ser un especial "limpio" o insignificante criterio; sin embargo, para aquellas personas que prefieren contar hiperplanos, es posible calcular N_t usando un método bastante similar a la forma que emplearemos para calcular ν_t.

3.4.2 Teoría de la prueba.

Para analizar el conjunto (2), inciamos observando que

$$\frac{1}{m}\, s^j(x) = \left\{ \frac{a^j x + (1 + a + \cdots + a^{j-1})c}{m} \right\} \bmod 1. \qquad (5)$$

Podemos omitir la operación "mod 1" extendiendo los conjuntos periódicamente, haciendo copias infinitamente de los hipercubos originales t-dimensionales, procediendo en todas direcciones. Esto nos da el conjunto

$$L=\left\{ \left(\frac{x}{m} + \underset{\sim}{k}_1, \frac{s(x)}{m}+k_2, \ldots, \frac{s^{t-1}(x)}{m}+k_t\right) \middle| x,k_1,k_2,\ldots,k_t \text{ son enteros} \right\}$$

$$=\left\{V_0 + \left(\frac{x}{m}+k_1, \frac{ax}{m}+k_2, \ldots, \frac{a^{t-1}x}{m}+k_t\right) \middle| x,k_1,k_2,\ldots,k_t \text{ son enteros}\right\}$$

donde

$$V_0 = \frac{1}{m}(0,c,(1+a)c,\ldots,(1+a+\cdots+a^{t-2})c) \qquad (6)$$

es un vector constante. La variable k_1 es redundante en su representaciön de L, debido que podemos cambiar $(x,k_1,k_2,\ldots,k_t)$ a $(x+k_1m,0,k_2 - ak_1,\ldots,k_1 - a^{t-1}k_1)$, reduciendo k_1 a cero sin perder generalidad. Por tanto podemos obtener la fórmula simple similarmente

$$L=\{V_0 + y_1V_1+y_2V_2 + \cdots + y_tV_t \mid y_1,y_2,\ldots,y_t \text{ son enteros }\}, \qquad (7)$$

donde

$$V_1 = \frac{1}{m}(1,a,a^2,\ldots,a^{t-1}); \qquad (8)$$

$$V_2 = (0,1,0,\ldots,0); \quad V_3 = (0,0,1,\ldots,0), \quad V_t = (0,0,0,\ldots,1) \qquad (9)$$

Los puntos $(x_1,x_2,\ldots,x_t)$de L que satisfacen $0 \le x_j < 1$ para toda j son precisamente todos los puntos de nuestro conjunto original (2).

Notemos que el incremento de e aparece sólo en V_0, y el efecto de V_0 es meramente cambiar todos los elementos de L sin cambiar sus distancias relativas; por lo tanto *e no afecta la prueba espectral* en cualquier forma, y podemos suponer que $V_0 = (0,0,\ldots,0)$ cuando calculemos ν_t. Cuando V_0 es el vector cero tenemos algo que llamamos *rejilla* de puntos

$$L_0=\left\{y_1V_1 + y_2V_2 + \cdots + y_tV_t \mid y_1,y_2,\ldots,y_t \text{ son enteros }\right\}, \qquad (10)$$

y nuestro objetivo es estudiar la distancia entre hiperplanos $(t-1)$-dimensionales adjuntos, en familias de hiperplanos paralelos que cubren todos los puntos de L_0.

Una familia de hiperplanos paralelos $(t-1)$-dimensionales

pueden ser definidos por un vector diferente de cero $U = (u_1,\ldots,u_t)$ que es perpendicular a todos ellos, y el conjunto de puntos en un hiperplano particular es entonces

$$\left\{(x_1,\ldots,x_t) \mid x_1u_1 + \cdots + x_tu_t = q\right\}, \tag{11}$$

donde q es una constante diferente para cada hiperplano en la familia. En otras palabras, cada hiperplano es el conjunto de toda X para la cual el *producto punto* $X\cdot U$ tiene un valor dado q. En nuestro caso los hiperplanos están todos separados por una distancia fija, y uno de ellos contiene $(0,0,\ldots,0)$; por lo tanto podemos ajustar la magnitud de U de modo que el conjunto de todos los valores *enteros* q dan todos los hiperplanos en la familia. Entonces la distancia entre hiperplanos vecinos es la distancia mínima de $(0,0,\ldots,0)$ al hiperplano para $q = 1$, denominado

$$\min_{x_1,\ldots,x_t \text{ reales}} \left\{\sqrt{x_1^2 + \cdots + x_t^2} \mid (x_1u_1 + \cdots + x_tu_t) = 1\right\}. \tag{12}$$

La desigualdad de Cauchy nos dice que

$$(x_1u_1 + \cdots + x_tu_t)^2 \le (x_1^2 + \cdots + x_t^2)(u_1^2 + \cdots + u_t^2), \tag{13}$$

por lo tanto el mínimo de (12) sucede cuando cada $x_j = u_j/(u_1^2 + \cdots + u_t^2)$; la distancia entre hiperplanos vecinos es

$$1/\sqrt{u_1^2 + \cdots + u_t^2} = 1/\text{long } (U). \tag{14}$$

En otras palabras, la cantidad ν_t que buscamos es precisamente la longitud del vector U más corto que define una familia de hiperplanos $\{X\cdot U = q \mid q$ es entero$\}$ que contienen todos los elementos de L_0.

Tal vector $U = (u_1,\ldots,u_t)$ debe ser diferente de cero, y debe satisfacer $V\cdot U$ = entero para toda U en L_0. En particular, puesto

que los puntos $(1,0,\ldots,0),(0,1,\ldots,0),\ldots,(0,0,\ldots,1)$ están todos en L_0, todos los u_i deberán ser enteros. Además puesto que u_t está en L_0, debe tener $\frac{1}{m}(u_1 + au_2 + \cdots + a^{t-1}u_t)$ = entero, es decir

$$u_1 + au_2 + \cdots a^{t-1}u_t \equiv 0 \text{ (modulo } m). \qquad (15)$$

Recíprocamente, cualquier vector entero diferente de cero $U = (u_1,..,u_t)$ que satisface (15) define una familia de hiperplanos requeridos, donde todos los puntos de L_0 estarán cubiertos: $(y_1v_1 + \cdots + y_tv_t)\cdot U$ será entero para todo entero $y_1,\ldots,y_t$. Hemos probado que

$$\nu_t^2 = \min_{(u_1,\ldots,u_t)\neq(0,\ldots,0)}\left\{u_1^2+\cdots+u_t^2 \,\middle|\, u_1+au_2+\cdots+a^{t-1}u_t\equiv 0 \text{ (modulo } m)\right\}$$

$$= \min_{(x_1,\ldots,x_t)\neq(0,\ldots,0)}((mx_1-ax_2-a^2x_3-\cdots-a^{t-1}x_t)^2+x_2^2+x_3^2+\cdots+x_t^2) \quad (16)$$

3.4.3 Deduciendo un método computacional.

Ahora hemos reducido la prueba espectral al problema de encontrar el valor mínimo (16); pero ¿cómo podemos determinar el valor mínimo en un tiempo razonable?. Una búsqueda super intensiva está fuera de la cuestión, entonces m es muy grande en casos de práctica interesante.

Será de interés y probablemente más útil si desarrollamos un método computacional para resolver aún problemas más generales: *Encontrar el valor mínimo de la cantidad.*

$$f(x_1,\ldots,x_t) = (u_{11}x_1+\cdots+u_{t1}x_t)^2+\cdots+(u_{1t}x_1+\cdots+u_{tt}x_t)^2 \qquad (17)$$

sobre todos los vectores *enteros diferentes de cero* $(x_1,\ldots,x_t)^2$, dada cualquier matriz no singular de coeficientes $U = (u_{ij})$. La expresión (17) es llamada una "forma cuadrática definida positiva" en t variables. Puesto que U es no singular (17) no puede ser cero

a menos que x_j sean todos cero.

Escribamos $u_1,\ldots,u_t$ para las filas de U. Entonces (17) puede escribirse como

$$f(x_1,\ldots,x_t) = (x_1u_1+\cdots+x_tu_t)\cdot(x_1u_1+\cdots+x_tu_t), \qquad (18)$$

el cuadrado de la longitud del vector $x_1U_1 + \cdots + x_tU_t$. La matriz no singular U tiene un inverso, lo cual significa que podemos encontrar únicamente determinados vectores $V_1,\ldots,V_t$ tales que

$$U_i\cdot V_j = \delta_{ij}, \qquad 1 \le i,j \le t. \qquad (19)$$

Por ejemplo en la forma especial (16) que surge de la prueba espectral, tenemos

$$\begin{aligned}
U_1 &= (\ \ m,0,0,\ldots,0) & V_1 &= \tfrac{1}{m}(1,a,a^2,\ldots,a^{t-1}),\\
U_2 &= (\ -a,1,0,\ldots,0) & V_2 &= (0,1,0,\ldots,\ \ 0),\\
U_3 &= (-a^2,0,1,\ldots,0) & V_3 &= (0,0,1,\ldots,\ \ 0),\\
&\quad\ldots\ldots\ldots\ldots & &\quad\ldots\ldots\ldots\ldots\\
U_t &= (-a^{t-1},0,0,\ldots,1) & V_t &= (0,0,0,\ldots,\ \ 1).
\end{aligned} \qquad (20)$$

Estos V_j son precisamente los vectores (8) y (9) que usamos para definir nuestra rejilla original L_0. Esto no es una coincidencia, desde luego si hemos comenzado con una rejilla arbitraria L_0, definido por cualquier conjunto de vectores linealmente independientes $V_1,\ldots,V_t$, el argumento que hemos usado arriba puede generalizarse para mostrar que la máxima separación entre hiperplanos en una familia es equivalente a minimizar (17), donde los coeficientes u_{ij} están definidos por (19).

Nuestro primer paso en la minimización de (18) es reducirla a un problema finito, es decir, mostrar que no necesitamos probar infinitamente muchos vectores $(x_1,\ldots,x_t)$ para encontrar el mínimo. Este está donde los vectores $V_1,\ldots,V_t$ se acercan; tenemos

$$x_k = (x_1U_1 + \cdots + x_tU_t)\cdot V_k,$$

y la desigualdad de Cauchy nos dice que

$$((x_1U_1 + \cdots + x_tU_t)\cdot V_k)^2 \le f(x_1,\ldots,x_t)(V_k\cdot V_k).$$

por lo tanto hemos deducido un límite superior útil en cada coordenada x_k.

Lema A. Sea $(x_1,\ldots,x_t)$ un vector no nulo que minimiza (18) y sea $(y_1,\ldots,y_t)$ cualquier vector entero diferente de cero. Entonces

$$x_k^2 \le (V_k\cdot V_k)f(y_1,\ldots,y_t) \quad \text{para } 1 \le k \le t, \tag{21}$$

en particular, siendo $y_i = \delta_{ij}$ para todo i,

$$x_k^2 \le (V_k\cdot V)(U_j\cdot U_j), \quad \text{para } 1 \le j,\ k \le t \tag{22}$$

El lema A reduce el problema a una búsqueda finita, pero el lado derecho de (21) usualmente es mucho más grande para hacer una búsqueda exhaustiva; necesitamos al menos una idea más. En tales ocasiones, una máxima antigua proporciona un sano consejo: "Si no puedes resolver un problema como está establecido, cámbialo a un problema más sencillo que tenga la misma respuesta". Por ejemplo, el algoritmo de Euclides tiene esta forma, si no conocemos el mcd del número de entradas, las cambiamos a números más pequeños que tienen el mismo mcd.

En nuestro caso, un problema más simple es uno que requiere menor búsqueda debido a que el lado derecho de (22) es más pequeño. La idea principal que usaremos es que es posible cambiar una forma cuadrática a otra que es equivalente para todo propósito práctico. Sea j cualquier valor fijo, $1 \le j \le t$; sea $(q_1,\ldots,q_{j-1},q_{j+1},\ldots,q_t)$ cualquier secuencia de $t-1$ enteros; y consideremos la siguiente transformación de los vectores

$$V_i' = V_i - q_iV_j, \quad x_i' = x_i - q_ix_j, \quad U_i' = U_i, \quad \text{para } i \ne j;$$

$$V_j = V_j, \qquad x'_j = x_j, \qquad U'_j = U_j + \sum_{i \neq j} q_i U_i. \tag{23}$$

es fácil ver que los nuevos vectores $U'_1,\ldots,U'_t$ definen una forma cuadrática f' para la cual $f'(x'_1,\ldots,x'_t) = f(x_1,\ldots,x_t)$; además la condición de ortogonalidad básica (19) sigue siendo válida, porque es fácil verificar que $U'_i \cdot V'_j = \delta_{ij}$. Como $(x_1,\ldots,x_t)$ recorre todos los vectores enteros diferentes de cero, así lo hace $(x'_1,\ldots,x'_t)$; por lo tanto la nueva forma f' tiene el mismo mínimo que f.

Nuestro objetivo es usar la transformación (23), remplazando U_i por U'_i y V_i por V'_i para toda i, para hacer el lado derecho de (22) pequeño, el lado derecho de (22) será pequeño cuando $U_j \cdot U_j$ y $V_k \cdot V_k$ sean pequeños, a la vez. Por tanto es natural plantear las dos siguientes preguntas acerca de la transformación (23),

a) ¿Qué elección de q_i hace $V'_i \cdot V'_i$ tan pequeño como sea posible?

b) ¿Cuál elección de $q_1,\ldots,q_{j-1},q_{j+1},\ldots,q_t$ hace $U'_j \cdot U'_j$ tan pequeño como sea posible?

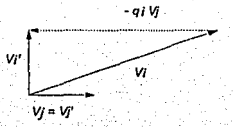
Es más fácil resolver estas preguntas para valores reales de q_i. La pregunta (a) es bastante sencilla, puesto que

$$\begin{aligned}(V_i - q_i V_j)\cdot(V_i - q_i V_j) &= V_i \cdot V_i - 2q_i V_i \cdot V_j + q_i^2 V_j \cdot V_j \\ &= (V_j \cdot V_j)(q_i - (V_i \cdot V_j / V_j \cdot V_j))^2 \\ &\quad + V_i V_i - (V_i V_j)^2 / V_j V_j,\end{aligned}$$

y el mínimo ocurre cuando

$$q_i = V_i \cdot V_j / V_j \cdot V_j. \tag{24}$$

Geométricamente, estamos preguntando que múltiplo de V_j debería sustraerse de V_i de modo que el vector resultante V'_i tenga longitud mínima, y la elección de q_i de manera que U'_i sea perpendicular a V_j (es decir, en forma que $V'_i \cdot V_j = 0$); el diagrama (25) muestra este plano.



(25)

Regresando a la pregunta (b), deseamos elegir q_i en forma que $U_j + \Sigma_{i \neq j} q_i U_i$ tenga longitud mínima; geométricamente, deseamos iniciar con U_j y agregar algún vector en el hiperplano $(t-1)$-dimensional cuyos puntos son la suma de múltiplos de $\{U_i \mid i \neq j\}$. De nuevo es elegir aquellos elementos de forma que U'_j sea perpendicular al hiperplano, es decir, en forma tal que $U'_j \cdot U_k = 0$ para toda $k \neq j$, es decir,

$$U_j \cdot U_k + \sum_{i \neq j} q_i (U_i \cdot U_k) = 0 \qquad 1 \leq k \leq t, \quad k \neq j. \tag{26}$$

Ahora que hemos dado respuesta a las preguntas (a) y (b), estamos en un momento de incertidumbre; ¿deberíamos elegir q_i de acuerdo a (24), en forma que $V'_i \cdot V'_i$ se minimice, o de acuerdo a (26), de manera que $U'_j \cdot U'_j$ se minimice? O de estas alternativas hacer una mejora del lado derecho de (22), así no es claro inmediatamente cuál elección obtendría prioridad. Afortunadamente, hay una respuesta muy simple para este dilema: Las Ocondiciones (24) y (26) son exactamente las mismas!. Por lo que las preguntas (a) y (b) tienen la misma respuesta, tenemos un momento feliz en el cual podemos reducir la longitud de U'_s y de V'_s simultáneamente.

Hasta aquí hemos tratado con las preguntas (a) y (b) sólo para valores *reales* de q_i. Nuestra aplicación nos restringe a valores enteros, así no podemos hacer V'_i exactamente perpendicular a U_j.

Lo mejor que podemos hacer para la pregunta (a) es asignar a q_i el entero *más cercano* a $V_i \cdot V_j / V_j \cdot V_j$ (25). Resulta que esta *no siempre* es la mejor solución para la pregunta (b), de hecho U'_j puede en ocasiones ser mayor que U_j. Sin embargo, el límite (21) nunca se incrementa, como podemos recordar el valor más pequeño de $f(Y_1,\ldots,Y_t)$ se encuentra hasta aquí. Esta elección de q_i basada sólo en la pregunta (a) es bastante satisfactoria.

Si aplicamos la transformación (23) repetidamente en tal forma que ninguno de los vectores V_i se obtiene más largo y al menos uno se obtiene más corto, nunca podremos entrar en un ciclo, es decir, nunca consideraremos la misma forma cuadrática de nuevo después para una secuencia de transformaciones no triviales de este tipo. Pero ocasionalmente conseguiremos "estancarnos", en el sentido de que ninguna de las transformaciones (23) para $i \leq j \leq t$ serán útiles para acortar cualquiera de los vectores $V_1,\ldots,V_t$. Al punto que podemos recurrir a una búsqueda exhaustiva, usando los límites del lema A, el cual será bastante pequeño en la mayoría de los casos. Eventualmente estos límites (21) serán pobres, y el algoritmo obtendrá otro tipo de transformación destrabándose otra vez y reduciendo los límites. Sin embargo, la transformación (23) por sí ha probado ser bastante adecuada para la prueba espectral; de hecho, ha probado ser asombrosamente poderosa cuando los cálculos son ordenados como en el algoritmo siguiente.

3.4.4 Como realizar la prueba espectral.

Ahora aquí está un procedimiento computacional eficiente que surge de nuestras consideraciones. R. W. Gasper y U. Dieter observaron que es posible usar los resultados de bajas dimensiones para realizar la prueba espectral significativamente más rápido en dimensiones altas. Este refinamiento se ha incorporado en el siguiente algoritmo, junto con una simplificación significante del caso bi-dimensional.

Algoritmo S.

Este algoritmo determina el valor de

$$\nu_t = \min\left\{\sqrt{x_1^2 + \cdots + x_t^2} \;\middle|\; x_1 + ax_2 + \cdots + a^{t-1}x_t \equiv 0 \pmod{m}\right\} \quad (27)$$

para $2 \le t \le T$, dado a, m, y T, donde $0 < a < m$ y a es primo relativo de m. (El número ν_t mide la exactitud t-dimensional de generadores de números aleatorios, como se vió anteriormente). Toda la aritmética en este algoritmo está hecha con enteros cuyas magnitudes exceden m^2, excepto en el paso S8; de hecho, aproximadamente todas las variables enteras serán menores que m en valor absoluto durante los cálculos.

Cuando ν_t está siendo calculada para $t \ge 3$, el algoritmo trabaja con dos matrices U y V de orden $t \times t$, estos vectores son denotados por $U_i = (u_{i1}, .., u_{it})$ y $V = (v_{i1}, \ldots, v_{ik})$ para $1 \le i \le t$. Estos vectores satisfacen las condiciones

$$u_{i1} + au_{i2} + \cdots + a^{t-1}u_{it} \equiv 0 \pmod{m}, \quad 1 \le i \le t; \quad (28)$$

$$U_i \cdot V_j = \delta_{ij} m, \quad 1 \le i, j \le t \quad (29)$$

(Esto es, V_j de nuestra discusión anterior ha sido multiplicado por m, para asegurar que sus componentes son enteros). Hay otros tres vectores auxiliares, $X = (x_1, \ldots, x_t)$, $Y = (y_1, \ldots, y_t)$, y $Z = (z_1, \ldots, z_t)$. Durante todo el algoritmo, r denotará $a^{t-1} \bmod m$ y s denotará al límite superior mínimo para ν_t^2 que hasta aquí se conoce.

S1. [Inicia] Inicializar $h = a$, $h' = m$, $p = 1$, $p' = 0$, $r = a$, $s = 1 + a^2$. (El primer paso de este algoritmo maneja el caso $t=2$ por un método especial, muy parecido al algoritmo de Euclides; tendremos $h = ap \equiv h' - ap' \equiv 0 \pmod{m}$ y $hp' - h'p = \pm m$ (30) durante esta fase del cálculo).

S2. [Paso euclideano] Inicializar $q = \lfloor h'/h \rfloor$, $u = h' - qh$, $v = p' - qp$. Si $u^2 + v^2 < s$, inicializar $s = u^2 + v^2$, $h' = h$, $h =$

u, $p' = p$, $p = v$, y repetir el paso S2.

S3. [Calcula ν_2] Hacer $u = u - h$, $v = v - p$; y si $u^2 + v^2 < s$, hacer $s = u^2 + v^2$, $h' = u$, $p' = v$. Entonces hacer $\sqrt{s} = \nu_2$ (Ahora construiremos la matrices U y V que satisfacen (28) y (29), en preparación de los cálculos para dimensiones más grandes). Hacer

$$U \leftarrow \begin{pmatrix} -h & p \\ -h' & p' \end{pmatrix}, \qquad V \leftarrow \pm \begin{pmatrix} p' & h' \\ -p & -h \end{pmatrix},$$

donde el signo - se elige para V si y sólo si $p' > 0$.

S4. [Avanza t] Si $t = T$, el algoritmo termina. (En cualquier otro caso deseamos incrementar t en 1. En este punto U y V son matrices $t \times t$ que satisfacen (28) y (29), y debemos ampliarlos agregando un nuevo renglón y una nueva columna apropiados). Hacer $t = t + 1$ y $r = (ar) \bmod m$. Hacer U_t el nuevo renglón $(-r,0,0,\ldots,0,1)$ de t elementos, y hacer $u_{it} = 0$ para $1 \le i \le t$. Hacer V_t el nuevo renglón $(0,0,0,\ldots,0,m)$. Finalmente, para $1 \le i \le t$, hacer $q = \text{round}(v_{it}r/m)$, $v_{it} = v_{it}r - qm$, y $U_t = U_t + qU_i$. (Aquí "round (x)" denota el entero más cercano a x, es decir, $\lfloor x + \frac{1}{2} \rfloor$. Esencialmente estamos inicializando $v_{it} = v_{it}r$ e inmediatamente aplicamos la transformación (23) con $j = t$, puesto que los números $|v_{it}r|$ son tan grandes que deben reducirse inmediatamente). Finalmente haciendo $s = \min(s, U_t \cdot U_t)$, $k = t$, y $j = 1$. (En los siguientes pasos, j denota el índice del renglón actual para la transformación (23), y k denota el último índice donde la transformación acorta al menos uno de los V_i).

S5. [Transformación] Para $1 \le i \le t$, hacer las siguientes operaciones: Si $i \ne j$ y $2|V_i \cdot V_j| > V_j \cdot V_j$ hacer $q = \text{round}(V_i \cdot V_j / V_j \cdot V_j)$, $V_i = V_i - qV_j$, $U_j = U_j + qU_i$, y $k = j$. El hecho de que omitamos esta transformación, cuando $2|V_i \cdot V_j|$ es exactamente igual a $V_i \cdot V_j$, advirtiendo al algoritmo de algún loop infinito).

S6. [Examinar nuevo límite] Si $k=j$ (es decir, la transformación en S5 ha sido de alguna utilidad), hacer $s = \min(s, U_j \cdot U_j)$.

S7. [Incrementar j] Si $j=t$, hacer $j = 1$; en otro caso hacer $j = j + 1$. Ahora si $j \neq k$, regresar al paso S5. (Si $j=k$, hemos ido hasta $t - 1$ ciclos consecutivos sin ninguna transformación, así el proceso de transformación está estancada).

S8. [Preparación para la búsqueda] (Ahora se determinará el mínimo absoluto, usando una búsqueda exhaustiva para toda $(x_1, \ldots, x_t)$ que satisface la condición (21) del lema A). Hacer $X = Y = (0, \ldots, 0)$, hacer $k = t$, y hacer

$$z_j = \left\lfloor \sqrt{\lfloor (V_j \cdot V_j) s/m^2 \rfloor} \right\rfloor, \quad \text{para } 1 \leq j \leq t \qquad (31)$$

(examinaremos todas las $X = (x_1, \ldots, x_t)$ con $|x_j| \leq z_j$ para $1 \leq j \leq t$. En cientos de aplicaciones de este algoritmo, ninguna z_j ha resultado ser mayor que 1, la búsqueda exhaustiva en los siguientes pasos aún no ha reducido s, sin embargo, tal fenómeno es posible probablemente en casos raros, especialmente en dimensiones mayores. Durante la búsqueda exhaustiva, el vector Y siempre será igual a $x_1 U_1 + \cdots + x_t U_t$, de forma que $f(x_1, \ldots, x_t) = Y \cdot Y$. Puesto que $f(-x_1, \ldots, -x_t) = f(x_1, \ldots, x_t)$, examinaremos sólo los vectores cuyo primer componente es positivo diferente de cero. El método es esencialmente este contador de pasos de uno, considerando $(x_1, \ldots, x_t)$ como los dígitos en un sistema numérico balanceado con raices mixtas $(2z_1 + 1, \ldots, 2z_t + 1)$).

S9. [Avanza x_k] Si $x_k = z_k$, ir a S11. En otro caso incrementar x_k por 1 y hacer $Y = Y + U_k$.

S10. [Incrementar k] Hacer $k = k + 1$. Entonces si $k \leq t$, hacer $x_k = -z_k$, $Y = Y - 2z_k U_k$, y repetir el paso 10. Pero si $k > t$, hacer $s = \min(s, Y \cdot Y)$.

S11. [Disminuir k] Hacer $k = k - 1$. Si $k \geq 1$, regresar a S9. En otro caso se da $\nu_t = \sqrt{s}$ (la búsqueda exhaustiva está

completa) y se regresa al paso S4.

En la práctica el algoritmo S se aplica para $T = 5$ ó 6, es decir, usualmente trabaja razonablemente bien cuando $T = 7$ u 8, pero puede ser terriblemente lento cuando $T \geq 9$ puesto que la búsqueda exhaustiva tiende a hacer el tiempo de corrida igual a $3T$. (Si el valor mínimo de ν_t ocurre en varios puntos diferentes, la búsqueda exhaustiva los encontraría todos, por lo tanto típicamente encontramos que toda $z_k = 1$ para t grande.Como señalamos arriba, los valores de ν_t son generalmente irrelevantes para propósitos prácticos cuando t es grande) (Knuth, 1969).

CAPITULO 4

ATRIBUTOS DE LOS ALGORITMOS GENERADORES DE NUMEROS ALEATORIOS SELECCIONADOS PARA EL ESTUDIO

4.1 DESCRIPCION DE ALGORITMOS

En los capítulos anteriores hemos tratado temas generales fundamentales en el análisis de un generador de números aleatorios, en el presenta capítulo presentamos siete algoritmos importantes en la generación de números aleatorios uniformes y se da la descripción de cada uno de ellos de acuerdo a lo expuesto por cada uno de los autores.

4.1.1 : Wichmann - Hill.

Este algoritmo hace uso del método Congruencial Multiplicativo. Utilizando el número primo 30,269 y el multiplicador 171. De manera que la regla generadora es:

$$X_{i+1} = (171 * X_i) \quad MOD \quad 30,269$$

lo que da una secuencia de valores para X_i a través de todos los valores entre 1 y 30,268 en un orden determinado, antes de repetirse la secuencia.

Una de las condiciones para obtener un buen generador es la portabilidad, en el caso del lenguaje Pascal, el generador quedaría escrito como:

$$X := (171 * X) \quad MOD \; 30269$$

En la práctica, este código no funcionará adecuadamente en todas las máquinas. Los valores de X se restringen al rango de 1 hasta 30,268 , ocupando una palabra de 16 bits. Esto no se cumple para $171 * X$, lo que provoca un error en muchas máquinas. Este cálculo se puede trabajar en " doble longitud ", pero, o se requiere código de máquina o es muy caro en tiempo y espacio. Si a X lo escribimos en la forma:

$$177 * r + s$$

donde

$$0 \leq s \leq 176$$

entonces:

$$171*X = 171*s+171*177*r = 171*s+30267*r = 171*s+30269*r-2*r,$$

como

$$30269*r \mod 30269 = 0,$$

$$(171*X) \mod 30269 = (171*s - 2*r) \mod 30269.$$

Por lo que, la sentencia de Pascal puede escribirse ahora como:

```
X := 171 * (X mod 177) - 2*(X DIV 177);
```

Ahora, para evitar la negatividad de X se agrega la siguiente sentencia:

```
if  X < 0  then  X := X + 30269;
```

Con este procedimiento, se ha obtenido un mecanismo para hacer un generador simple para una máquina de 16 bits. Se elige un número primo ligeramente menor a 32,768 y un multiplicador (aproximadamente la raíz cuadrada del número primo).

Para mejorar el período y las propiedades estadísticas de este

generador, se utiliza la técnica de combinar dos o más generadores para obtener uno que es mejor estadísticamente, a cualquiera de sus componentes. Lo anterior se basa en que: si se consideran dos números aleatorios X_1 y X_2 en el rango $0 < X < 1$; llamamos la parte fraccionaria de X_1+X_2 la combinación de X_1+X_2. Es claro que si X_1 y X_2 son independientes y uniformemente distribuidos, entonces la combinación de X_1+X_2 está también uniformemente distribuida, sobre el mismo rango de valores. Por otra parte, si X_1 y X_2 es aproximadamente aleatorio, la combinación de X_1+X_2 será mucho más aproximada que cualquiera de las dos.

Para obtener el generador definitivo se han combinado tres generadores simples, su combinación se maneja correctamente sólo si son independientes. De hecho, no pueden ser completamente independientes, puesto que, si los tres primos son p, q y r, la longitud de las secuencias son $p-1$, $q-1$ y $r-1$; estos tienen necesariamente el 2 como factor común. Sin embargo, ellos pueden ser lo suficientemente independientes para propósitos prácticos, si no tiene ningún otro factor en común. De acuerdo a esto la longitud del ciclo del generador combinado es: $(p-1)(q-1)(r-1)/4$, que en nuestro caso es aproximadamente: 6.95E+12.

Pruebas aplicadas al algoritmo.

Primero se aplicó una prueba de series en las salidas del generador. Es sabido que un generador de un solo elemento fallará esta prueba, puesto que cualquier valor pequeño siempre es seguido por el mismo valor multiplicado por el multiplicador.

La segunda prueba consistió en simular manos de póker. Se realizaron cinco llamadas del generador para formar una mano. Se acumularon el número de manos con todas las cartas diferentes, un par, dos pares, y así sucesivamente.

La tercer prueba fue la de corridas arriba y abajo. Las pruebas para corridas arriba no son independientes de las pruebas para corridas abajo, así que se realizaron en forma separada. Se probaron los primeros cinco dígitos en la forma especificada por Grafton (Wichmann,Hill 1987 (Grafton, R.G.T.)).

En general, el generador pasó exitoso. Pero, de hecho, incluye defectos aproximadamente en la proporción adecuada de casos. Bien, no exactamente en la proporción adecuada, esto sería demasiado bueno para ser verdad, lo cual de nuevo sería una falla para un comportamiento aleatorio (Wichmann & Hill, 1987).

4.1.2 : Payne

Un algoritmo generalizado de retroalimentación para cambios de registro

Las ventajas de este algoritmo sobre cualquier otro generador de números pseudoaleatorios son las siguientes:

1. Produce números pseudoaleatorios multidimensionales.
2. La longitud del periodo es arbitraria, independiente del tamaño de palabra de la máquina en la que opere.
3. Es más rápido que otros generadores de números pseudoaleatorios.
4. Puede obtenerse la misma secuencia de números pseudoaleatorios en cualquier máquina; esto es, los bits de alto orden de la mantisa de cada número pseudoaleatorio concuerda en todas las máquinas: por ejemplo las series de computadora IBM 360, SPERRY-RAND-UNIVAC 1108, CONTROL DATA 6000 , y HEWLETT-PACKARD 2100.
5. Este puede ser codificado en lenguajes compilador (es portátil). Este generador de números aleatorios es ideal para mini y microcomputadoras, puesto que se necesitan para su ejecución sólo dos adiciones y un or-exclusivo.

El generador multiplicativo congruencial Lehmer de números aleatorios se ha probado para tener un espacio n no uniforme. Esta deficiencia es severa para máquinas de tamaño de palabra pequeña. Como una alternativa el generador de números aleatorios (RNG) de retroalimentacion de cambios de registros (FSR) pretende ser uniforme en un espacio n si palabras de p-bit son divididas en

n subpalabras.

Los generadores FSR con generación polinomial primitiva, X^p+X^q+1, con q pequeña o cercana a (p-1)/2, debe evitarse a causa de las propiedades de malas corridas. Sin embargo, la cuidadosa selección de p, q proporciona números aleatorios satisfactorios en un espacio de baja dimensión.

Tal vez las secuencias FSR ofrecen la mejor perspectiva para mejorar en espacio n. El algoritmo de Kendall es moderadamente rápido en la mayoría de las máquinas, pero el periodo es fijo para el tamaño de palabra y es difícil implementarlo en multiprecisión. Además, la división para obtener mayor uniformidad en el espacio n recorta la longitud del ciclo y la resolución. Este problema es intrínseco a las secuencias periódicas. Una secuencia cíclica de m números, cuando se toma en pares, localiza sólo m de m^2 puntos en un enrejado bidimensional de m por m. En general, el enrejado de puntos $m^n m$ nunca se localizará en el espacio n por n-tuples tomados de una secuencia periodica m, es decir, la mayoría de los productos cruzados están perdidos. Esta escases en un espacio n sostiene no uniformidad de todos los RNGs periódicos. Aparentemente, lo que se necesita es un RNG que permita repetir números dentro de una secuencia de periodo completo. Tales números repetidos podrían llenar en puntos m^n, para algún n.

Definimos $\oplus$ como el operador 'or-exclusivo' que es equivalente a la adición módulo 2. El algoritmo de Kendall se usa para seleccionar sucesivas n-tuples de la secuencia básica (a_i), donde

$$a_k = a_{k-p+q} \oplus a_{k-p}\ ,\ k=p;\ p+1;\ \ldots,\ \text{dado}\ a_{p-1},\ldots,a_0$$

y el cambio de registro de retroalimentación basado en un polinomio primitivo X^p+X^q+1. Por ejemplo, X^5+X^2+1 y $a_0=a_1=a_2=a_3=a_4=1$ produce $(a_i)_0^{30}$= 11111000110111010100000100101100. Seleccionando 5-tuples, A_i, por el algoritmo de Kendall produce los números aleatorios vistos en la figura 4.1.

Es importante observar que cada A_i (con excepción de 0) ocurre una vez y sólo una vez en el periodo completo de una secuencia aleatoria Tausworthe (la 'secuencia Kendall' es una secuencia

Tausworthe específica).

La idea bajo el algoritmo generalizado de retroalimentación de cambios de registro de números aleatorios (GFSR) es que la secuencia básica de cambios de registros (a_j) basado en el trinomio primitivo X^p+X^q+1 se inicializa en j columnas, j<p, con un retraso, cuidadosamente seleccionado, entre columnas. Por ejemplo, si elegimos de nuevo el trinomio primitivo x^5+x^2+1 . La

A_0	1 1 1 1 1	A_{10}	0 1 0 0 0	A_{20}	1 0 1 1 0
A_1	1 1 0 0 0	A_{11}	1 1 0 1 0	A_{21}	0 1 0 1 1
A_2	0 1 1 1 0	A_{12}	1 1 1 0 0	A_{22}	0 0 0 0 1
A_3	0 0 1 0 1	A_{13}	0 0 0 1 1	A_{23}	0 1 0 0 1
A_4	0 0 1 0 0	A_{14}	1 1 0 1 1	A_{24}	1 0 0 1 1
A_5	0 1 1 0 1	A_{15}	1 0 1 0 1	A_{25}	0 1 1 1 1
A_6	1 1 1 1 0	A_{16}	1 0 0 0 0	A_{26}	0 1 1 0 0
A_7	1 0 0 0 1	A_{17}	1 0 1 0 0	A_{27}	1 0 1 1 1
A_8	1 1 1 0 1	A_{18}	1 1 0 0 1	A_{28}	0 0 0 1 0
A_9	0 1 0 1 0	A_{19}	0 0 1 1 1	A_{29}	1 0 0 1 0
				A_{30}	0 0 1 1 0

FIGURA 4.1
"secuencia Kendall" para el polinomio x^5+x^2+1

secuencia básica de (a_j) se copia en la primer columna de la figura 4.2. Para este polinomio en particular la segunda columna se formó retrasando la primer columna por 25 (25-31=-6, dependiendo de la orientación del 'círculo' de bits) posiciones bit; la tercera columna se obtuvo 'retrasando' la segunda columna por otras -6 posiciones bit; y así sucesivamente hasta que se han llenado las 5 columnas.

Puesto que cada columna obedece a la recurrencia $a_k=a_{k-p+q} \oplus a_{k-p}$, cada palabra debe también obedecer $w_k=w_{k-p+q} \oplus w_{k-p}$.

Observando cuidadosamente cada w_j ocurre una y solo una vez en el período completo de $2^5-1 = 31$ números.

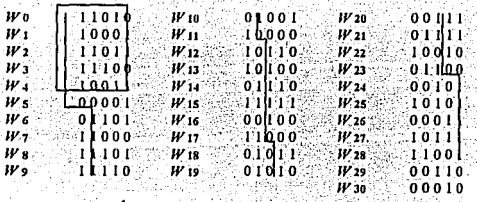


FIGURA 4.2

Secuencia GFSR para el polinomio $X^5 + X^2 + 1$ con retrasos de -6

Haciendo retrasos arbitrarios entre las columnas ¿se asegura que cada número entre 1 y 2^P-1 ocurre una y sólo una vez en cada periodo? No, pero alguna 'matriz inicial' o de inicialización (W_0, W_1, W_2, W_3, W_4 para X^5+X^2+1) puede verificarse para ver si esta contiene la propiedad deseable, y segundo, el número de secuencias diferentes con esta propiedad puede ser analíticamente calculado.

Por claridad proseguimos, utilizando el ejemplo de la figura 4.2. La matriz asociada al polinomio X^5+X^2+1 es :

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

así

$$
\begin{matrix} W_0 & W_1 & W_2 & W_3 & W_4 \end{matrix}
\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}
\quad
\begin{matrix} & & & C & & \end{matrix}
\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}
=
\begin{matrix} W_1 & W_2 & W_3 & W_4 & W_5 \end{matrix}
\begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix}
$$

o en la notación matricial $WoC=W_1$. Después de $2^5-1 = 31$ aplicaciones de esta matriz de recurrencia, $WoC^{31} = Wo$ o C^{31} es congruente módulo 2 a la matriz identidad. Claramente C, C^2, C^3, ... , y C^{30} son todas diferentes por virtud del periodo de la secuencia de cambios de registro. Esto es, para todo WoC^k, $1<k<30$, es diferente, sólo es necesario que Wo tenga filas linealmemte independientes. Esto se prueba fácilmente.

El número de secuencias diferentes que pueden producirse por el algoritmo GFSR es igual al número de formas diferentes de Wo con filas linealmente independientes que pueden seleccionarse.

Asignando r_1, r_2, r_3, r_4, y r_5 para denotar las cinco filas de la matriz Wo. La fila r_1 puede seleccionarse para ser alguno de los $2^5-1 = 31$ vectores binarios diferentes de cero. La fila r_2 puede ser alguno de los $2^5-1-1=30$ vectores binarios que incluye r_1 y el vector cero. La fila r_3 puede ser alguno de los $2^5-1-3=28$ vectores binarios que incluyen el vector cero y alguna combinación lineal de r_1 y r_2. La fila r_4 puede ser alguno de los $2^5-1-7=24$ vectores binarios excluyendo el vector cero y combinaciones lineales de r_1, r_2, r_3. El mismo argumento puede aplicarse a la selección de r_4. Esto es, con el algoritmo GFSR, es posible producir $(2^5-1)(2^5-2)(2^5-4)(2^5-8)(2^5-16) = (31)(30)(28)(24)(16) = 9,999,360$ secuencias diferentes basadas en el polinomio primitivo X^5+X^2+1.

El algoritmo GFSR es,

0. Si k ≠ 0 , ir a 2 (k inicialmente es cero).
1. Inicializar, $W_0, \ldots, W_{p-1}$ usando una secuencia básica de retraso (a_1) para obtener cada columna de $W_0, \ldots, W_{p-1} \cdot -1$.
2. K = k+1.
3. Si k>p, k = 1 iniciar.

4. j = k+q.
5. Si j>p, iniciar j = j-p.
6. or-exclusivo, $w_k \otimes w_j$.
7. Almacenar, $w_k \otimes w_j \quad w_k$.

Se observa una tabla de rotación de p palabras en GFSR, que se implementa como una función en PASCAL , algunos compiladores no pueden implementar operaciones lógicas de palabra completa, u otros pueden optimizar cálculos lógicos. Sin embargo, los números aleatorios GFSR pueden verificarse para validar de nuevo los resultados.

A. Inicialización del algoritmo GFSR.

El algoritmo GFSR se autoinicializa en el sentido de que las réplicas de retraso se producen por el mismo procedimiento que genera palabras completas. La independencia lineal de las columnas iniciales se garantizan si la medida de retraso máxima desde la columna de más a la izquierda es menor que el periodo completo, 2^p-1 (si la "constante de retraso" entre cada columna es primo relativo de 2^p-1, entonces el retraso máximo puede exceder el periodo completo). Usando este procedimiento, cada p-tuple se genera (excepto todos los ceros) antes de repetir algún p-tuple. Cada columna inicial es un p-tuple y por eso debe ser independiente de todas las otras. Por ejemplo, en la figura 4.2 la inicialización puede hacerse desde el más significante a los bits menos significantes (izquierda a derecha), iniciando con [11111]. La relación de recurrencia se aplica 25 veces, aquí, se obtiene la siguiente columna [10110]. Unas segundas 25 aplicaciones dan como resultado [00010], unas terceras 25 aplicaciones resultan en [10101]; y finalmente, se obtiene [01101]. La recurrencia se aplica llamando el GFSR RNG con ceros colocados a la derecha. Cada nueva columna se cambia a la derecha después de generarse, y la columna original [11111] se remplaza a la columna del extremo izquierdo. En generadores más reales (tamaño de palabra grande), el periodo completo no se agotará an la inicialización cumpliendo

con las condiciones de independencia lineal (aquí, se garantizó la independencia lineal porque 25 es primo relativo de 31).

Para facilitar la implementación, se usa la inicialización p-tuple [111...] en SETR. Sin embargo, el SETR aplica 5000p retrasos adicionales así que la columna iniciada [111...] esta "recurrenciada", también da un patron aleatorio de bits iniciales antes que todos los unos. Esta "repetición" adicional es necesaria para permitir desaparecer lentamente [111...]. Los efectos de tal patron regular aplazan los posteriore p-tuples. Por ejemplo, [111...], p=98, q=27, se transforma en 72 ceros, 26 unos; siguen 45 ceros, 53 unos, etc.. Las agrupaciones de todos los ceros y todos los unos llegan a ser progresivamente más pequeños y más aleatorios a través de aplicaciones repetidas de la relación de recurrencia. La "extinción" de la p-tuple inicial se hace en SETR aplicando la relación de recurrencia 5000p veces a valores iniciales de palabra completa. Debe notarse que si no se desea la generalidad relativa al tamaño de palabra, entonces la "repetición" adicional no es necesaria cuando la inicialización se realiza desde el bit menos significante al más significante. El bit más significante ha sido retrasado p DELAY veces, después se "repite" una vez por cada bit en una palabra p-bit (el bit menos significante es uno). Esto es, la "aleatoriedad" adicional del patrón inicial [111...] se ha realizado sin trabajo adicional.

Finalmente, SETR regresa un valor para el número de columnas linealmente independientes disponibles al procedimiento de inicialización.

B. Generalidad del GFSR

El paralelo natural de GFSR inmediatamente generaliza a máquinas L-bit (tamaño entero) independientes de la relación entre L y p, esto es, para L<p, ocurrirán muchos números repetidos, pero la longitud del ciclo, m, es aún 2^p-1. El caso donde L=3 y x^5+x^2+1, de la figura 4.2, se demuestra en la figura 4.3 . Aquí, 2^{p-L} duplicados diferentes de cero y $2^{p-L}-1$ ceros se producen en un periodo completo.

W_0	1 1 0	W_{10}	0 1 0	W_{20}	0 0 1
W_1	1 0 0	W_{11}	1 0 0	W_{21}	0 1 1
W_2	1 1 0	W_{12}	1 0 1	W_{22}	1 0 0
W_3	1 1 1	W_{13}	1 0 1	W_{23}	0 1 1
W_4	1 0 0	W_{14}	0 1 1	W_{24}	0 0 1
W_5	0 0 0	W_{15}	1 1 1	W_{25}	1 0 1
W_6	0 1 1	W_{16}	0 0 1	W_{26}	0 0 0
W_7	0 1 0	W_{17}	1 1 0	W_{27}	1 0 1
W_8	1 1 1	W_{18}	0 1 0	W_{28}	1 1 0
W_9	1 1 1	W_{19}	0 1 0	W_{29}	0 0 1
				W_{30}	0 0 0

FIGURA 4.3
Algoritmo GFSR para L=3, polinomio x^5+x^2+1

C. Periodo del GFSR.

Es posible un periodo "ilimitado" sin incrementar el tamaño de palabra de la mayoría de las máquinas. Por ejemplo, $M=2^{532}-1$ se obtiene usando $X^{532}+X^{37}+1$ de la tabla en la figura 4.4. Para agotar este ciclo requeriría muchos años en una computadora muy rápida, es decir, si se generaran 10^6 números/seg., aproximadamente se necesitarían 10^{150} años para completar el ciclo. Es más importante para el autor, la repetibilidad de números en un periodo completo.

D. Pruebas aplicadas al algoritmo.

Se realizaron las siguientes pruebas empíricas en 10,000 datos obtenidos por el GFSR usando $X^{98}+X^{27}+1$, 15-bits, y la estadística chi-cuadrada.

La prueba de frecuencias contó el número de datos que cayeron en cada una de las 100 celdas iguales en el intervalo (0,1).

p	q
47	5,14,20,21
95	11,17
98	11,27
111	10,49
124	37
170	23
250	103
380	47
476	15,141
532	37

FIGURA 4.4

Polinomios primitivos, x^p+x^q+1, p grande

La prueba de intervalos cuenta la longitud de intervalos entre dígitos sucesivos escalando el número aleatorio normalizado a dígitos decimales. La prueba se aplica a dígitos entre 0 y 9.

La prueba de autocorrelación calcula el máximo coeficiente de autocorrelación normalizado al intervalo 50. La correlación aceptable está en el intervalo (0.03,0.08).

La prueba D^2 compara la distribución teórica de una línea aleatoria en dos dimensiones con la distribución obtenida empíricamente.

La prueba de series cuenta el número de pares de datos que caen en 100 celdas iguales. Aquí, los pares de dígitos decimales se combinan para dar un entero de 0 a 99. Se aplicó una prueba de frecuencias a las 100 celdas.

La prueba de corridas calcula la corrida más larga, compara la frecuencia de cada corrida con la distribución teórica, y el número de corridas arriba/abajo de la media. El número total de corridas se compara con el número esperado.

La prueba del mínimo/máximo de n compara la distribución empírica y teórica de los números mínimo/máximo de n. Se cuenta el número de los extremos para cada una de las 100 celdas y se aplica una prueba de frecuencias para $n = 2,4,6,\ldots,20$.

La prueba bit condicional cuenta el número de bits uno en la j-ésima posición dados los j-1 bit previos. Se forma un árbol binario con una rama para cada posición de bit. El árbol de bits enumerado se compara con el valor esperado de $(\frac{1}{2})*10,000 = 5000$. Si los bits son de verdad independientes, entonces el valor empírico iguala el valor esperado.

La prueba de la tranformada finita de Fourier (FFT) prueba para un plano espectral las estadísticas

$$U = (S - \tfrac{1}{2})\sqrt{(12M)}, \qquad P_n = \sum_{r=1}^{n} |a_n|^2 \Big/ \sum_{r=1}^{MM} |a_n|^2, \qquad n = 1,2,\ldots,M+1,$$

$$S = \frac{1}{M}\sum_{n=0}^{M} P_n, \qquad M = \begin{cases} N/2, & N \text{ par}, \\ (N-3)/2, & N \text{ impar}, \end{cases}$$

y la secuencia transformada

$$A_n = \frac{1}{N}\sum_{j=0}^{N-1} R_j \exp(-2\pi i jn/N); \qquad n = 0,1,\ldots,N-1;$$

R_j = número aleatorio.

La simulación de la dispersión del experimento compara la distribución esperada de aspectos de dispersión con los valores teóricos esperados. Un punto de la superficie de una esfera se elige aleatoriamente esto da un aspecto denso. Este aspecto denso representa la desviación de un neutrón después de chocar con un átomo. El aspecto denso es inverso al peso de acuerdo a $\sigma(\theta)=\pi(1-\theta/\pi): 0\leq\theta\leq\pi$. La distribución empírica de θ se compara con la teórica usando una prueba chi-cuadrada con 36 intervalos en $(0,\pi)$.

El algoritmo GFSR para 15 bits se ejecutó satisfactoriamente en las pruebas mencionadas arriba, al nivel de 5 porciento de significancia. Se notaron fallas con n = 8,14,20 en la prueba del mínimo de n. Sin embargo, no se detectó ninguna falla para n = 10,12,16,18 y toda n probada en la prueba máximo de n. Esto es, en esta prueba se da una indicación parcial de no uniformidad para $n>6$.

El algoritmo GFSR puede ser considerado estadísticamente calificado como un RNG superior en velocidad y generalidad a otros contemporáneos RNG's (Lewis & Payne, 1973).

4.1.3 : Kaplan (a)

Este es un generador multiplicativo para una computadora de 32 bits.

Para la ejecución de este algoritmo debe inicializarse la semilla NEWSED con un valor impar, ya que esta es argumento de valor del procedimiento RINIT.

El valor de MULT es bastante grande, cercano al máximo valor que puede almacenar una palabra de computadora. En el proceso de la multiplicación de SEED anterior por MULT, muchos bits se propagan a la izquierda del espacio disponible para su registro, una situación llamada *overflow* (desbordamiento), pero en nuestro caso, se descarta intensionalmente la parte más informativa del producto. La parte restante, el orden más bajo de 32 bits tiene la menor información acerca de la magnitud del anterior SEED, aún cuando este se determinó completamente por los bits del SEED anterior.

Debido a la parte del producto que se conserva, para todo propósito práctico, no relacionando la magnitud de SEED anterior, podemos usar la secuencia de productos generados de esta manera, efectivamente como: no correlacionados, magnitudes distribuidas uniformemente: necesitamos escalar todos los valores al rango de 0 a 1, dividiendo entre 2^{31} y tomando el valor absoluto (Kaplan, 1981).

Pruebas aplicadas al algoritmo

El autor no presenta información sobre pruebas de aleatoriedad realizadas para las secuencias de números aleatorios generadas por este algoritmo, por lo que hemos aplicado la prueba de promedios y la de frecuencias para sustentar con esta herramienta (pruebas de

aleatoriedad) la uniformidad de las secuencias generadas por el algoritmo.

La prueba de promedios. Se probó la hipótesis de que los números siguen un comportamiento uniforme con media = 0.5, con un intervalo de confianza del 95% obteniéndose que la hipótesis no se puede rechazar (figura 4.5).

La prueba de frecuencias. Para una secuencia con N=1000 y n=30 intervalos, con intervalo de confianza del 95% donde $X^2_{0.95,29}$ = 42.56; se obtuvo el estadístico X^2_0 = 26.12. Como $X^2_0 < X^2_{0.95,29}$ la hipótesis no puede ser rechazada.

Número de observaciones : 1000

Promedio	0.505972
Varianza	0.0862682
Desviación estandar	0.293714
Mediana	0.524359

Intervalo de confianza para la media: 95%
Intervalo 0.487742 , 0.524203 999 g.l.

Prueba de hipótesis para

H0: Media = 0.5 Estadístico t calculado = 0.643007
vs H1: Media ≠ 0.5 Nivel de significancia = 0.520367
con α = 0.05 no se rechaza H0.

FIGURA 4.5
Resultados de la prueba de promedios

4.1.4 : Kaplan (b)

Para computadoras con hardware sin restricciones de multiplicación o con instrucciones de multiplicación que no conservan bits de bajo orden después del *overflow*, no es conveniente utilizar el método congruencial multiplicativo; en su lugar se emplea un generador que se basa en una lista circular de números de semilla, circular en el sentido de que el primer artículo sigue lógicamente al último. Para generar un entero aleatorio, el entero aleatorio anterior se agrega al siguiente artículo y se usa en el entero aleatorio actual; entonces se escalan a un número real entre 0 y 1.

Este generador no sólo proporciona una semilla de 32 bits, sino que proporciona una lista completa de ellos, donde al menos uno es impar. Al contrario de la transformación extremadamente compleja de bits dada por el generador multiplicativo, este generador aplica una operación aditiva guardando la magnitud del número actual y un número de la distancia relativa anterior, de este modo se logra un buen grado de no correlación entre valores sucesivos. La lista de 22 no es arbitraria, se basa en la existencia de ciertas propiedades de polinomios de grado 22, asegurando que la secuencia alcanza la longitud máxima antes de repetirse. En este generador la longitud de palabra tiene muy poco efecto en las propiedades estadísticas (Kaplan, 1981).

Pruebas aplicadas al algoritmo

Al igual que para el algoritmo anterior, no se tiene información sobre pruebas de aleatoriedad realizadas para las secuencias de números aleatorios generadas,por lo que de la misma manera, hemos aplicado la prueba de promedios y la prueba de frecuencias para comprobar en forma estadística la uniformidad de las secuencias generadas por el algoritmo.

La prueba de promedios. Se probó la hipótesis de que los

números siguen un comportamiento uniforme con media = 0.5, con un intervalo de confianza del 95% obteniéndose que la hipótesis no se puede rechazar (figura 4.6).

La prueba de frecuencias. Para una secuencia con N=1000 y n=30 intervalos, con intervalo de confianza del 95% donde $X^2_{0.95,29}$ = 42.56; se obtuvo el estadístico X^2_0 = 23.54. Como $X^2_0 < X^2_{0.95,29}$ la hipótesis no puede ser rechazada.

Número de observaciones : 1000

Promedio	0.507076
Varianza	0.0847899
Desviación estandar	0.291187
Mediana	0.505416

Intervalo de confianza para la media: 95%
Intervalo 0.489003 , 0.52515 999 g.l.

Prueba de hipótesis para

H0: Media = 0.5 Estadístico t calculado = 0.768479
vs H1: Media ≠ 0.5 Nivel de significancia = 0.442384
con α = 0.05 no se rechaza H0.

FIGURA 4.6

Resultados de la prueba de promedios

4.1.5 : RANECU
(l'Ecuyer)

Un generador está definido por un espacio muestral finito S, una función $f:S \rightarrow S$ y un estado inicial S_0 llamado la semilla. El estado del generador se desarrolla de acuerdo a la recursión

$$s_i := f(s_{i-1}) \; ; \qquad i = 1,2,3,... \tag{1}$$

y el estado actual s_i en la etapa i usualmente se transforma en un valor real entre 0 y 1, de acuerdo a

$$U_i := g(s_i) \tag{2}$$

donde $g:S \rightarrow (0,1)$. El periodo del generador es el entero más pequeño p tal que

$$s_{i+p} = s_i \qquad \text{para toda} \quad i>v \tag{3}$$

para algún entero $v \geq 0$.

Es bien aceptado que para obtener un buen generador, la elección de f y g deberá estar basada en un firme respaldo teórico, y antes de ser usado para aplicaciones prácticas, el generador deberá someterse a un considerado conjunto de pruebas teóricas. Una buena implementación del generador debería ser razonablemente rápida, portátil y usar poca memoria en la computadora.

El generador empleado más comunmente hoy en día es el *generador congruencial lineal* (GCL) de Lehmer, para el cual

$$f(s) = (as + c) \bmod m; \qquad g(s) = s/m; \tag{4}$$

donde el *modulo* m y el *multiplicador* $a<m$ son enteros positivos; y la constante $c<m$ es un entero no negativo. Usualmente se elige

c=0, en cuyo caso el generador se llama *generador congruencial multiplicativo lineal* (GCML) y su *espacio muestral* es S = {1,2,...,m-1}. El GCLM tiene período máximo (p = m-1) si m es primo y a es un elemento módulo m.

El algoritmo presentado por l'Ecuyer es una combinación de GCMLs y cada uno de ellos satisface las restricciones teóricas para la calidad de la secuencia que genera. Este método de generación es nuevo y produce un generador cuyo período es el mínimo común múltiplo de los periodos individuales.

La mejoría *matemáticamente demostrada* de los generadores combinados sobre sus componentes es un periodo mucho más grande. Más allá de esto, la combinación es un intuitivo atractivo eurístico respaldado por las pruebas empíricas y el hecho de que ciertas patologías demostrables en los componentes no son aparentes en la combinación.

El método para combinar varios GCMLs y obtener uno mejor se basa en los lemas siguientes. El lema 1 generaliza un comentario informal planteado por Wichmann y Hill.

LEMA 1. Sean $W_1,\ldots,W_l$ variables aleatorias discretas independientes tal que W_1 es uniforme entre 0 y d-1, donde d es un entero positivo:

$$Pr(W_1 = n) = \frac{1}{d} \tag{5}$$

Entonces

$$W = \left(\sum_{j=1}^{l} W_j \right) \mod d \tag{6}$$

sigue una ley de probabilidad uniforme discreta entre 0 y d-1.

DEMOSTRACION. Primero mostramos los resultados para l=2. Sea W_2 discreta (no necesariamente uniforme) entre a y b. Para $0 \le n \le d-1$, tenemos :

$$Pr(W=n) = \sum_{k=0}^{\infty} Pr(W_1+W_2 = n + kd)$$

$$= \sum_{i=a}^{b} Pr(W_2 = i)\ Pr(W_1 = (n-i) \bmod d)$$

$$= \frac{1}{d} \sum_{i=a}^{b} Pr(W_2 = i) = \frac{1}{d}\ .$$

En palabras, cualquiera que sea el valor de i tomado por W_2, $W=(W_1+W_2) \bmod d$ es uniforme entre 0 y $d-1$.

Para $l>2$, se definen las variables aleatorias:

$$V_2 = (W_1 + W_2) \bmod d$$
$$V_3 = (V_2 + W_3) \bmod d$$
$$\vdots$$
$$V_l = (V_{l-1} + W_l) \bmod d.$$

El resultado para $l=2$ implica que todos aquellos V_j son discretos uniformes entre 0 y $d-1$ puesto que $W=V_l$, esto completa la demostración. ∎

LEMA 2. Considere una familia de l generadores donde, para $j=1,\ldots,l$, el generador j tiene un periodo p_j y se desarrolla de acuerdo a

$$s_{j,i} := f_j(s_{j,i-1}) \qquad (7)$$

Entonces el periodo p de la secuencia $\{s_i = (s_{1,i},\ldots,s_{l,i},\ i = 0,1,2,\ldots\}$, donde $s_0=(s_{1,0},\ldots,s_{l,0})$ es una semilla dada, es el mínimo común múltiplo de $p_1,\ldots,p_l$.

DEMOSTRACION. Cada generador individual j tiene periodo p_j, así que p es un múltiplo de p_j para cada j. Si algún entero n es un múltiplo de cada p_j, entonces claramente $s_{i+n} = s_i$ para cualquier $i\geq 0$, lo cual implica que $p\leq n$. ∎

Ahora consideremos al caso donde cada generador individual j es un GCML de periodo máximo con módulo m_j y multiplicador a_j :

$$s_{j,i} = f_j(s_{j,i-1}) = a_j s_{j,i-1} \bmod m_j \qquad (8)$$

Combinamos estos generadores como sugiere la ecuación (6) con $d=m_1-1$. El generador j tiene periodo m_j-1 donde m_j es primo. Por tanto, cada $p_j = m_j-1$ pueden igualarse y así un límite superior para p está dado por:

$$p \leq \frac{\prod_{j=1}^{l} (m_j - 1)}{2^{l-1}} \qquad (9)$$

y este límite se obtiene sólo cuando todos los valores de $(m_j-1)/2$ son relativamente primos. Durante un ciclo completo, el generador (1) toma cada valor $1,\ldots,m-1$ sólo una vez. Esto es, estipulando que el generador (1) es bastante bueno, $s_{1,i}-1$ puede considerarse como una variable discreta uniforme entre 0 y m_1-2 por la ecuación (5). En adelante, suponemos que $s_{1,i},\ldots,s_{l,i}$ son variables aleatorias independientes con $s_{1,i}$ uniforme en $\{1,\ldots,m_1-1\}$. De acuerdo al lema 1,

$$Z_i = \left(\sum_{j=1}^{l} (-1)^{j-1} s_{j,i} \right) \bmod (m_1-1) \qquad (10)$$

es como una variable eleatoria discreta uniforme entre 0 y m_1-2 y

$$U_i = \begin{cases} Z_i/m & \text{si } Z_i > 0 \\ (m_1-1)/m_1 & \text{si } Z_i = 0 \end{cases} \qquad (11)$$

es por tanto una buena aproximación de una variable aleatoria continua uniforme (0,1) para m_1 bastante grande.

Sólo Z_1 necesita ser uniforme para que el lema 1 sea válido. Pero en la práctica, Z_1 no es exactamente uniforme. Por tanto, es eurísticamente más atractivo para tener todas las Z_i tan uniformes como sea posible.

A. Presentación de dos nuevos generadores

De lo expuesto anteriormente y de los resultados obtenidos en la búsqueda de implementacines portátiles, a continuación se proponen dos nuevos generadores combinados. Para una longitud de palabra de b-bit, deseamos $m_j < 2^{b-1}$ y $a_j \leq \sqrt{m_j}$ para todo j, de manera que cada generador individual puede implementarse en una forma portátil y eficiente usando la técnica propuesta por Bratley y Wichmann.

Para computadoras de 32-bit, sugerimos l=2, m_1=2147483563, a_1=40014, m_2=2147473399 y a_2=40692. Estos dos GCMLs individualmente son excelentes de acuerdo a la prueba espectral. Además, $(m_1-1)/2 = 3\times7\times631\times81031$ y $(m_2-1)/2 = 19\times31\times1019\times1789$ son primos relativos y el generador combinado tiene periodo $p = (m_1-1)(m_2-1)/2 = 2.30584\times10^{18}$.

Para computadoras de 16-bit, sugerimos l=3 y seleccionamos los tres GCMLs definidos por m_1=32363, a_1=157, m_2=31727, a_2=146, m_3=31657 y a_3=142. Todos ellos presentan muy bien la prueba espectral y los valores de $(m_1-1)/2 = 11\times1471$, $(m_2-1)/2 = 29\times547$ y $(m_3-1)/2 = 2\times2\times3\times1319$ son primos relativos. El periodo del generador combinado entonces es $p = (m_1-1)(m_2-1)(m_3-1)/4 \approx 8.12544\times10^{12}$.

El algoritmo 4.1 da una función del primer generador combinado propuesto y el algoritmo 4.2 da la función para el otro generador combinado propuesto para computadoras de 16-bits.

B. Pruebas aplicadas al algoritmo.

El generador MLCG se sometió a un comprensible número de pruebas estadísticas descritas en Knuth (1969). Cada prueba produce un estadístico que, bajo la hipótesis nula Ho que el generador es bueno, tiene una distribución de probabilidad teórica conocida. Además, cada prueba se ha repetido N veces y la distribución empírica de los valores de los estadísticos se ha comparado con la distribución teórica usando la clásica prueba de Kolmogorov-Smirnov (KS). Esto es, el resultado final es el valor s

de un estadístico KS S. Un generador falla la prueba si el nivel descriptivo observado $\delta = Pr(S \geq s | H_0)$ es "demasiado pequeño".

Se realizaron 21 pruebas diferentes . Estas se describen usando la notación de Knuth para sus parámetros. Aquí, n denota el número de observaciones durante una corrida dada y N denota el número de corridas. Estas pruebas involucran billones de números pseudoaleatorios y tomaron más de 200 en tiempo de CPU en una VAX-11/780.

(1) Prueba de equidistribución, usando chi-cuadrada, d=64, n=1000, N=10000.
(2) Prueba de equidistribución, usando chi-cuadrada, d=256, n=10000, N=10000.
(3) Prueba serial con pares (2-dimensional), d=64, n=100000, N=1000.
(4) Prueba serial con triples (3-dimensional), d=16, n=100000, N=1000.
(5) Prueba serial con cuadruples (4-dimensional), d=8, n=100000, N=1000.
(6) Prueba de intervalos, α=0, β=0.05, t=15, n=10000, N=1000.
(7) Prueba de intervalos, α=0.95, β=1, t=15, n=10000, N=1000.
(8) Prueba de intervalos, α=1/3, β=2/3, t=10, n=10000, N=1000.
(9) Prueba de póker, k=4, d=4, n=10000, N=1000.
(10) Prueba de póker, k=6, d=4, n=10000, N=1000.
(11) Prueba de póker, k=6, d=8, n=10000, N=1000.
(12) Prueba de póker, k=8, d=16, n=10000, N=1000.
(13) Prueba de colector de cupones, d=5, t=25, n=10000, N=1000.
(14) Prueba de colector de cupones, d=10, t=40, n=10000, N=1000.
(15) Prueba de permutaciones, t=3, n=10000, N=1000.
(16) Prueba de permutaciones, t=5, n=10000, N=1000.
(17) Prueba de corridas arriba, n=100000, N=1000.
(18) Prueba del máximo de t, t=8, d=128, n=10000, N=1000.
(19) Prueba de colisiones, 6 dimensiones, d=8, n=20000, N=100.
(20) Prueba de colisiones, 10 dimensiones, d=4, n=20000, N=100.
(21) Prueba de colisiones, 20 dimensiones, d=2, n=20000, N=100.

Los resultados de las pruebas aparecen en la tabla 4.1 . Las semillas iniciales para cada prueba fueron ISEED1=12345 y ISEED2=67890 (l 'Ecuyer, 1988).

prueba	δ
1	0.0961
2	0.7984
3	0.7388
4	0.4399
5	0.7530
6	0.8818
7	0.0751
8	0.1881
9	0.1879
10	0.6358
11	0.3925
12	0.3395
13	0.9390
14	0.4053
15	0.8859
16	0.3516
17	0.1775
18	0.8703
19	0.9341
20	0.2101
21	0.1019

TABLA 4.1
Resultados de la prueba empírica

```
FUNCTION Uniform : REAL;
     VAR
          Z,k : INTEGER;
     BEGIN
     k   := s1 DIV 53668;
     s1  := 40014 * (s1 - k * 53668) - k * 12211;
     IF s1 < 0 THEN s1 := s1 + 2147483563;

     k   := s2 DIV 52774;
     s2  := 40692 * (s2 - K * 52774) - k * 3791;
     IF s2 < 0 THEN s2 := s2 + 2147483399;

     Z := s1 - s2;
     IF Z < 1 THEN Z := Z + 2147483562;

     Uniform := Z * 4.656613E-10
     END
```

ALGORITMO 4.1
Generador portable para computadoras de 32 bits

```
FUNCTION Uniform : REAL;
     VAR
          Z,k : INTEGER;
     BEGIN
     k   := s1 DIV 206;
     s1  := 157 * (s1 - k * 206) - k * 21;
     IF s1 < 0 THEN s1 := s1 + 32363;

     k   := s2 DIV 217;
     s2  := 146 * (s2 - K * 217) - k * 45;
     IF s2 < 0 THEN s2 := s2 + 31727;

     k   := s3 DIV 222;
     s3  := 142 * (s3 - K * 222) - k * 133;
     IF s3 < 0 THEN s3 := s3 + 31657;


     Z := s1 - s2;
     IF Z > 706 THEN Z := Z - 32362;
     Z := Z + s3;
     IF Z < 1 THEN Z := Z + 32362;

     Uniform := Z * 3.0899E-5
     END
```

ALGORITMO 4.2
Generador portable para computadoras de 16 bits

4.1.6 : RANMAR

(Marsaglia - Zaman - Tsang)

Este generador, es el primero de una nueva generación de métodos portátiles VLP (very long period). Tiene un periodo de $2^{144} \simeq 2\times10^{43}$, es completamente portátil, da resultados de bits idénticos en una máquina con mantisa de al menos 24 bits en la representación del punto flotante, (es decir, todas las computadoras comunes de 32 bits o más). Es bastante rápido, en gran parte porque trabaja internamente, en representación de punto flotante.

La propiedad más excepcional de este generador es la extrema simplicidad de generar secuencias independientes. El generador debe ser inicializado dando un entero de 32 bits, *cada valor da origen a una secuencia independiente de longitud suficiente para un cálculo entero.* Esto es, el programa puede generar aproximadamente 900 millones de secuencias diferentes, cada una muy grande (promedio de longitud $\simeq 10^{30}$).

El algoritmo es una combinación de una secuencia de Fibonacci (con intervalos de 97 y 33, y operación "substracción plus uno, módulo uno"). La "secuencia aritmética", no es usado en muchos generadores estándar, pero resultó ser bastante bueno cuando se combinó con otro método, como la secuencia de Fibonacci con retrasos, el cual es ya bastante bueno. La combinación de las dos secuencias está hecha además con la operación "substracción plus uno, módulo uno", y todas las operaciones están realizadas fuera en punto flotante asumiendo una mantisa de al menos 24 bits. La tabla inicial de 97 valores esta inicializada utilizando una combinación de un método de retrasos de Fibonacci usando 3 atrasos, y un método lineal congruencial con a=53 y m=169.

La versión original de Marsaglia y Zaman es igual al que se presenta, excepto que ahora se proporciona un arreglo de números en lugar de uno.

La rutina de inicialización también se modificó ligeramente. La

original requiere de inicializar con 4 números enteros pequeños, ahora se requiere de sólo uno, entre 0 y 900 millones (James, 1990).

Pruebas aplicadas al algoritmo

Las propiedades más importantes del generador se resumen el la siguiente tabla. La unidad "1 wd" significa una palabra de 32-bit.

aleatoriedad	portabilidad	periodo aprox.	requiere para iniciar [wd]	requiere para reiniciar [wd]	secuencias independientes no. x long.
buena	buena	10^{43}	1	100	$10^{9} \times 10^{34}$

Complementando la información proporcionada por el autor, se aplicaron dos pruebas de hipótesis a una secuencia de 1000 números generados por el algoritmo, la prueba de promedios y la prueba de frecuencias.

La prueba de promedios. Se probó la hipótesis de que los números generados siguen un comportamiento uniforme con media = 0.5, con un intervalo de confianza del 95% obteniéndose que la hipótesis no se puede rechazar (figura 4.7).

La prueba de frecuencias. Para una secuencia con N=1000 y n=30 intervalos, con intervalo de confianza del 95% donde $X^2_{0.95,29}$ = 42.56; se obtuvo el estadístico X^2_0 = 33.26. Dado que $X^2_0 < X^2_{0.95,29}$ la hipótesis no puede rechazarse.

```
Número de observaciones : 1000
  Promedio                0.495072
  Varianza                0.0816008
  Desviación estandar     0.285659
  Mediana                 0.502317
  Intervalo de confianza para la media:   95%
  Intervalo      0.477342 ; 0.512803    999 g.l.

Prueba de hipótesis para

     H0: Media = 0.5       Estadístico t calculado = -0.545508
  vs H1: Media ≠ 0.5       Nivel de significancia = 0.585526
         con α = 0.05  no se rechaza H0.
```

FIGURA 4.7
Resultados de la prueba de promedios

4.1.7 : ACARRY

(Marsaglia y Zaman)

Los esfuerzos más recientes de Marsaglia y hermanos es una clase completa de generadores VLP conocidos como *sumar y llevar*. Se hace referencia a esta clase de generadores generalmente como ACARRY, pero la variación particular que se propone aquí es actualmente *substracción y pedir prestado,* y el nombre de la subrutina propuesta es RCARRY. El algoritmo es muy semejante al método de Fibonacci, pero con el agregado de un bit extra, de acuerdo a si la suma de Fibonacci fue mayor que uno. La fórmula básica es:

$$X_n = (X_{n-r} \pm X_{n-s} \pm c) \bmod b$$

donde $r>s$ son los retrasos, y c es un bit agregado, igual a

cero a menos que la suma fuese mayor que b, este está en uno de los bit menos significativos. El tamaño de palabra b puede elegirse igual a 2, en cuyo caso genera bits aleatorios, o un número más largo (usualmente una potencia de dos) para generar números más largos. En el ejemplo propuesto, se consideró $b=2^{24}$ para generar números de 32 bits en punto flotante con mantisas de 24 bits.

Como en el método de Fibonacci, r y s se eligieron de un conjunto de números para los cuales se conoce que el método produce un periodo muy largo. Y este algoritmo también requiere acumular las r semillas anteriores. Para $b=2^{24}$, una elección conveniente es r=24 y s=10, lo que da un periodo sólo de un factor de 48 más pequeño que el número de diferentes estados que pueden representarse por 24 números de 24 ·bits, lo que es $(2^{24})^{24}$. Esto es, el periodo para el generador dado es $\approx 2^{570}$ ó $\approx 10^{171}$. El estado del generador en algún momento puede especificarse por los valores de 24 enteros de 24 bits mas dos enteros pequeños y el valor del bit de acarreo, el cual puede ser asignado fácilmente en una 25^{va} palabra.

Resumiendo, es necesario inicializar el vector de 24 semillas (punto flotante) así como los dos índices I24 y J24, como en RMARIN, y el valor de RCARRY también debe inicializarse, pero puede ser iniciado con cero. La inicialización normal es o por default (lo que producirá siempre la misma secuencia) o por introducción de un entero (24 bits o menos) de los cuales cada valor inicia una secuencia muy larga ($\approx 10^{160}$) la cual no coincidirá con ninguna otra (James, 1990).

Pruebas aplicadas al algoritmo

Las propiedades más importantes del generador se resumen el la siguiente tabla. La unidad "1 wd" significa una palabra de 32-bit.

aleatoriedad	portabilidad	periodo aprox.	requiere para iniciar [wd]	requiere para reiniciar [wd]	secuencias independientes no. x long.
buena	buena	10^{170}	1	25	$10^{9} \times 10^{161}$

En la misma forma que en el algoritmo anterior hemos complementando la información proporcionada por el autor, aplicando dos pruebas de uniformidad a una secuencia de 1000 números generados por el algoritmo, la de promedios y la de frecuencias.

La prueba de promedios. Se probó la hipótesis de que los números generados tienen un comportamiento uniforme con media = 0.5, usando un intervalo de confianza del 95% se obtuvo que la hipótesis no puede rechazarse (figura 4.8).

La prueba de frecuencias. Para una secuencia con N=1000 y n=30 intervalos, con intervalo de confianza del 95% donde $X^2_{0.95,29}$ = 42.56; se obtuvo el estadístico X^2_0 = 27.14. Dado que $X^2_0 < X^2_{0.95,29}$ la hipótesis no puede ser rechazada.

```
Número de observaciones : 1000
  Promedio                0.496503
  Varianza                0.0810151
  Desviación estandar     0.284632
  Mediana                 0.514004
  Intervalo de confianza para la media:   95%
  Intervalo      0.478836 , 0.514169     999 g.l.

Prueba de hipótesis para

     H0: Media = 0.5       Estadístico t calculado = -0.388534
  vs H1: Media ≠ 0.5       Nivel de significancia = 0.697704
         con α = 0.05   no se rechaza H0.
```

FIGURA 4.8
Resultados de la prueba de promedios

4.2 CARACTERISTICAS

Como se ha señalado en el capitulo 2, un buen generador de números aleatorios debe generar secuencias que correspondan a una distribución uniforme, que ocupe un espacio mínimo en memoria, que sea rápido y que tenga periodo grande, que sea capaz de repetir la misma secuencia y que sea portable.

De acuerdo a la información proporcionada por los autores de los algoritmos presentados aquí, cada generador satisface cada una de estas características en forma satisfactoria.

El comportamiento uniforme de las secuencias ha sido probado por los autores, utilizando para ello algunas de las pruebas que ya hemos mencionado en el capítulo anterior, para visualizar esto se ha agregado a la programación de los algoritmos un módulo de graficación, donde se muestra a través de un diagrama de barras la frecuencia de los números aleatorios generados, de manera que podamos ver gráficamente las variaciones en la uniformidad de las secuencias generadas. De esta forma se advierte que la mayor variación de comportamiento la presenta el algoritmo de Wichmann y Hill, y en el resto de las secuencias generadas por los otros algoritmos se detectan ligeras variaciones con respecto a la distribución uniforme.

Por lo que se refiere al espacio en memoria, cinco de los generadores tienen como base el método congruencial de Lehmer, que por su regla recursiva ocupa un mínimo de memoria en la computadora; la velocidad de ejecución, es bastante buena y aproximadamente la misma en todos, de acuerdo a la información proporcionada por los autores, el mayor periodo lo tine el generador RCARRY y este es de 10^{160} y el mínimo es el de RINITMULT Y RINITADIT con 2.14748×10^{9}, el calificativo aplicado a la periodicidad depende de la aplicación que deseemos darle al generador.

Con todos estos algoritmos podemos volver a obtener la secuencia excepto en el de cambios de registro, que tiene una inicialización aleatoria; y el código en todos es fácilmente portable de una máquina a otra.

4.3 DESVENTAJAS

En general, las secuencias generadas por los algoritmos presentados tienen características muy satisfactorias de aleatoriedad, y son muy similares en cuanto a velocidad, portabilidad y en requerimientos de memoria.

El algoritmo de Wichmann y Hill es el que presenta mayor variación en su distribución, y en el tiempo de generación es ligeramente mayor al de los otros algoritmos.

Todos ellos permiten repetir la secuencia de números a excepción del algoritmo de Payne, que se inicializa con valores aleatorios y por ello la secuencia generada es diferente en cada ocasión que se utiliza el algoritmo, en este caso esta es la desventaja que se le observa.

CAPITULO 5

EVALUACION DE LOS ALGORITMOS

En el presente capítulo se realiza una evaluación de los siete algoritmos tratados en esta investigación; después de haber estudiado cada uno de ellos, identificando los métodos y características que los definen.

Para llegar a la fase de evaluación de los algoritmos, se ha trabajado sobre cada uno de ellos cuidadosamente, identificando sus bases, el método respectivo que utilizan los autores y sus características; todo esto basado en la descripción que da cada autor sobre su propio algoritmo. Con estos elementos se ha tratado de identificar las desventajas que como generadores de números aleatorios pudieran presentar. En el capítulo anterior ya hemos hablado sobre estos aspectos.

Un algoritmo es una serie de pasos ordenados que nos ayudan a resolver un problema, para poder realizar cualquier actividad del tipo que ésta sea es necesario llevar a cabo un conjunto de acciones que nos permitan al final de ellas ver concluida la actividad deseada, para obtener un generador de números aleatorios que además sea un buen generador, todos estos pasos deben en su conjunto, satisfacer los siguientes aspectos:

1.Generar secuencias de números que correspondan a una distribución uniforme y que sean estadísticamente independientes,

2. que ocupe un espacio mínimo en memoria de la computadora,

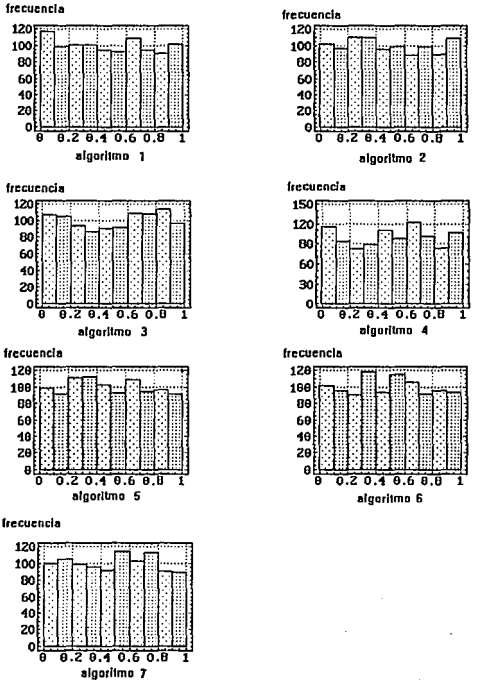
3. el tiempo de generación de un número debe ser el menor posible,

4. que el periodo de la secuencia sea grande y

5. debe permitir obtener la misma secuencia de números para poder repetir el experimento.

En nuestro trabajo de investigación analizamos siete de los algoritmos clásicos para generar números aleatorios. Como punto

DISTRIBUCION DE FRECUENCIA DE LAS SECUENCIAS GENERADAS POR LOS ALGORITMOS



GRAFICA 5.1

Las secuencias obtenidas presentan una uniformidad en los datos muy aceptable; la secuencia 4 tiene una mayor variación, estadísticamente pequeña ya que satisface la prueba de bondad de ajuste. Los siete algoritmos seleccionados, de acuerdo a esta información son muy buenos generadores de números aleatorios.

final de este análisis llevamos a cabo una evaluación de los algoritmos que como se presenta en el objetivo de la investigación consiste en identificar sus deficiencias respecto a las características que debe satisfacer para ser un buen generador de números aleatorios, nos damos cuenta de que cada uno de ellos cumple de manera satisfactoria con dichas características y que entre ellos no existen diferencias significativas desde el punto de vista estadístico y computacional; esto significa que son igualmente confiables y eficientes.

Las secuencias generadas por los siete algoritmos, de acuerdo a las pruebas estadísticas aplicadas por los autores, tienen un comportamiento uniforme; para observarlo gráficamente se generó una secuencia de datos con cada algoritmo y se obtuvo su respectiva gráfica en el paquete estadístico STATGRAPHICS (gráfica 5.1); el algoritmo de Kaplan (b) presenta un mayor nivel de variación en su distribución pero no la afecta significativamente. El espacio de memoria que ocupan cada uno de ellos es realmente pequeño, así como el tiempo que requieren para generar un número aleatorio. El periodo de las secuencias es grande y los algoritmos son fácilmente portables.

Esto nos permite afirmar que las secuencias generadas por los siete algoritmos tienen características satisfactorias de aleatoriedad, velocidad, portabilidad y de espacio en memoria. La tabla 5.1 (a) y (b) muestra los resultados del análisis estadístico.

Unicamente el algoritmo de Wichmann y Hill es el que ligeramente presenta mayor variación en su distribución y en cuanto al tiempo de generación, 6 de ellos tienen periodo muy similar, aunque se distingue el algoritmo de Marsaglia-Zaman-Tsang por presentar un tiempo un poco menor al de los otros generadores (gráfica 5.2). Para identificar un punto de evaluación diferente al estadístico ó computacional se realizó una entrevista con algunos investigadores que utilizan los números aleatorios. Teniendo como puntos de la entrevista el área donde aplican los números aleatorios, el método que utilizan para obtenerlos y las características que para su caso en particular exigen en un

Número de observaciones = 1000

	Algoritmo 5: L-Ecuyer	Algoritmo 6: Mar-Zam-Tsang	Algoritmo 7: Marsaglia-Zaman
Promedio	0.493597	0.495072	0.496503
Varianza	0.0815349	0.0816008	0.0810151
Desviación estandar	0.285543	0.285659	0.284632
Mediana	0.486625	0.502317	0.514004
Intervalo de Confianza para la media: 95 %			
L.I.	0.475874	0.477342	0.478836
L.S.	0.51132	0.512803	0.514169
con 999 g.l			
Prueba de hipotesis para: H0: Media = 0.5 vs H1: Media ' 0.5 al nivel a = 0.05			
estadistico t calculado:	-0.70908	-0.545508	-0.388534
nivel de significancia :	0.478441	0.585526	0.697704
decision :	no se rechaza H0.	no se rechaza H0.	no se rechaza H0.

TABLA 5.1 (B)

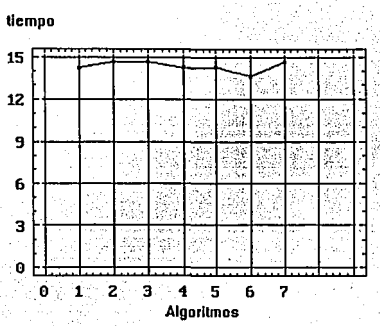
Se generó una secuencia de 1000 números aleatorios con cada uno de los generadores y con ayuda del paquete estadístico STATGRAPHICS se realizó el análisis de los datos, obteniéndose la tabla 5.1 (a) y (b), la cual nos muestra las medidas estadísticas más importantes. Se hizo una prueba de hipótesis para probar que la media de los valores generados en el intervalo (0,1) tienen una distribución uniforme con una media de 0.5, con un intervalo de confianza del 95% la hipótesis no se rechaza en ningún caso, lo que significa que las secuencias tienen una distribución uniforme, como se ha venido afirmando.

ANALISIS ESTADISTICO PARA LAS SECUENCIAS
DE NUMEROS ALEATORIOS

Número de observaciones = 1000

	Algoritmo 1: Wichmann-Hill	Algoritmo 2: Payne	Algoritmo 3: Kaplan (a)	Algoritmo 4: Kaplan (b)
Promedio	0.488034	0.49199	0.505972	0.499479
Varianza	0.0868991	0.0846985	0.0862682	0.0843466
Desviación estandar	0.294787	0.29103	0.293714	0.290425
Mediana	0.485549	0.484664	0.524359	0.514453
Intervalo de Confianza para la media: 95 %				
L.I.	0.469737	0.475926	0.487742	0.481452
L.S.	0.506331	0.512054	0.524203	0.517505
con 999 g.l				
Prueba de hipotesis para: H0: Media = 0.5 vs H1: Media ' 0.5 al nivel a = 0.05				
estadistico t calculado:	-1.28367	-0.65305	0.643007	-0.0567739
nivel de significancia :	0.199555	0.513875	0.520367	0.954737
decision :	no se rechaza H0.	no se rechaza H0..	no se rechaza H0.	no se rechaza

TABLA 5.1 (a)



GRAFICA 5.2

La gráfica muestra los tiempos de generación (centésimas de segundo) para una secuencia de 1000 números aleatorios de cada uno de los generadores.
Los tiempos de generación son muy parecidos entre sí; la graficación de ellos nos muestra uniformidad, de entre los siete generadores se destaca el generador número 6 que presenta un tiempo un poco menor al de los otros.

generador de números aleatorios. La información que nos proporcionaron nos lleva a afirmar que en la actualidad, debido al avance tecnológico, los generadores de números aleatorios que ofrece el mercado de software son altamente confiables; por lo que el investigador los utiliza directamente, sólo en aplicaciones muy especiales se detiene a analizar la periodicídad que ofrece el generador, por ejemplo en la simulación browniana. Actualmente, presenta un mayor punto de interés la generación de números aleatorios con distribuciones diferentes a partir de una secuencia aleatoria uniforme.

El software desarrollado a la par de esta investigación contiene una sección donde se genera una secuencia aleatoria uniforme con cualquiera de los siete algoritmos estudiados, y otra donde se pueden obtener secuencias aleatorias con cinco de las distribuciones más importantes:

1. **Binomial**, esta distribución es muy útil en las áreas de inspección de calidad, ventas, mercadotecnia, medicina e investigación de opiniones entre otras.

2. distribución de **Poisson**, en esta la variable aleatoria representa eventos aleatorios que ocurren de manera independiente con una velocidad constante en el tiempo o en el espacio, algunos ejemplos típicos son el número de personas que llegan a una tienda de autoservicio en un tiempo determinado, el número de defectos en piezas similares para el material, el número de bacterias en un cultivo, el número de solicitudes de seguro procesadas por una compañía en un periodo específico, etc, esta distribución es principalmente un modelo de probabilidad para analizar problemas de líneas de espera.

3. distribución **Normal**, esta distribución es muy importante en la aplicación de la inferencia estadística en el análisis de datos, gran número de estudios indican que proporciona una adecuada representación de las distribuciones de una gran cantidad de variables físicas, por ejemplo en datos metereológicos como la temperatura y la precipitación pluvial, mediciones efectuadas en organismos vivos, calificaciones en pruebas de actitud, mediciones físicas en partes manufacturadas, errores de instrumentación y

otras desviaciones de las normas establecidas, etc.

4. distribución Uniforme, esta distribución representa eventos que toman valores dentro de un intervalo infinito, de manera que estos se encuentran distribuidos igualmente sobre el intervalo.

5. distribución Weibull, esta distribución se usa básicamente en el área de confiabilidad para modelar la relación entre distintos componentes de materiales.

En conclusión, se presenta el paquete de computación que consta de dos módulos, uno que nos permite generar una secuencia de números aleatorios con distribución uniforme a partir de siete de los algoritmos clásicos utilizados para ello y otro donde se generan variables aleatorias con distribución Binomial, Poisson, Normal, Uniforme ó Weibull. En el primer módulo el usuario tendrá opción para proporcionar la semilla, si elige esta opción tendrá acceso a los algoritmos que tienen esta flexibilidad, de no desear esta opción la secuencia se generará con el algoritmo que tenga la semilla establecida. Si se introduce la semilla el paquete dará al usuario el periodo máximo alcanzado por el generador con esa semilla y la secuencia de valores aleatorios si es que al usuario le satisface el periodo. Una vez generados los valores, el usuario podrá ver el diagrama de barras que representa la frecuencia de los datos generados y podrá escribir los valores en archivo o en papel.

En el segundo módulo se genera la secuencia de números aleatorios a partir de uno de los siete algoritmos del primer módulo, dejando la posibilidad de que sea seleccionado por el usuario. La salida de los datos puede hacerse por archivo o solo por pantalla.

CONCLUSIONES

Los generadores de números aleatorios de Wichmann y Hill, Payne, Kaplan, l'Ecuyer, Marsaglia, Zaman y Tsang presentados en este trabajo, de acuerdo a los estudios que estos investigadores han realizado, muestran aleatoriedad y uniformidad en las secuencias que generan; en forma adicional y una vez programados estos algoritmos, se obtuvo una secuencia de cada uno de ellos y se aplicó una prueba de promedios por lo que pudo comprobarse directamente que las secuencias se ajustan a una función de distribución uniforme. Se generó una secuencia de números para cada uno de los algoritmos tomando el tiempo de proceso, resultando que las diferencias en el tiempo de generación son muy pequeñas, siendo ligeramente más rápido el algoritmo de Marsaglia-Zaman-Tsang. Estos resultados reflejan la confiabilidad ofrecida por los siete algoritmos, que pueden ser utilizados en el desarrollo de aplicaciones con la seguridad de que proporcionan números aleatorios que cumplen satisfactoriamente con los requerimientos estadísticos y computacionales.

Si bien es cierto que debido a los avances en el área de la computación podemos contar con calculadoras que tienen integrada una pequeña función generadora de números aleatorios, y con paquetes estadísticos que generan números aleatorios uniformes o con alguna otra función de distribución, de probabilidad el material desarrollado durante esta investigación es una herramienta que ofrece la misma confiabilidad que aquellos y podrá ser más accesible para los estudiantes que lo requieran por ser propiedad de la universidad.

Existen investigaciones cuyos estudios emplean números aleatorios con una función diferente a la uniforme, pero la generación de números aleatorios uniformes no pierde su importancia debido a que para generar cualquier distribución de

probabilidad es necesario contar con un generador de números aleatorios uniformes y una función que a través de un método de transformación nos permita generar la distribución deseada.

Considerando esta necesidad, el programa que se ha desarrollado contiene un módulo donde se generan números aleatorios con las distribuciones consideradas entre las más importantes para la investigación: Binomial, Poisson, Normal y Weibull, incluyendo la distribución uniforme para un intervalo (a,b); de forma que puedan utilizarse en las investigaciones que requieran de ello.

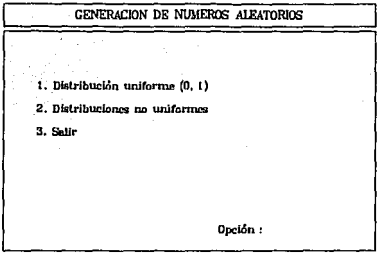
ANEXOS

RANDOM

Este programa de computación contiene los siete algoritmos analizados en el trabajo. El material ofrece la posibilidad de generar números aleatorios y almacenarlos en archivo de datos o ,en caso de generarse una cantidad pequeña de ellos, pueden verse en pantalla únicamente. Posteriormente se presenta el diagrama de frecuencias de los valores de la secuencia.

Dada la gran importancia que también tiene la generación de números aleatorios con distribución no uniforme se incluye la generación de números aleatorios con las distribuciones de probabilidad: Binomial, Poisson, Normal, Weibull y la Uniforme en un intervalo (a,b).

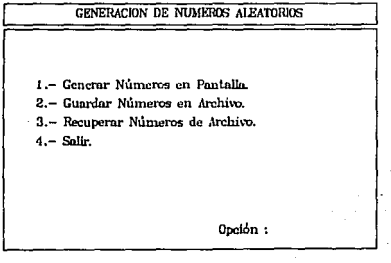
Pantalla No.1



Se tiene acceso a dos importantes secciones: la primera se refiere a los siete algoritmos que generan números aleatorios uniformes en el intervalo (0,1); y la segunda, a la generación de números aleatorios con distribuciones de probabilidad no uniformes; estas son cinco distribuciones que se encuentran entre las más importantes. utilizadas en las áreas de investigación: Binomial, Poisson, Normal, Weibull, y Uniforme

Seleccionando la opción 1 o 2 la siguiente pantalla será la número 2. En todas las pantallas la opción salir implica abandonar el programa.

Pantalla No.2



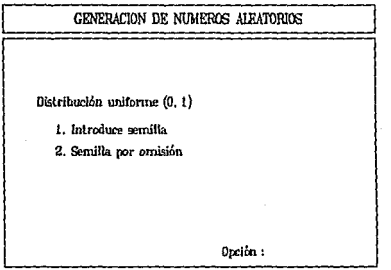
Los números generados pueden ser almacenados en un archivo de datos o pueden ser presentados sólo en la pantalla. Si en alguna ocasión anterior se almacenaron en archivo estos pueden recuperarse y verse en pantalla y si fueron generados

con cualquiera de los siete algoritmos generadores de números uniformes puede verse su diagrama de frecuencias.

Si seleccionamos la opción 1 o 2 la siguiente pantalla será la número 3. La opción 3 significa que tenemos datos almacenados en disco y tendremos a continuación la pantalla número 8.

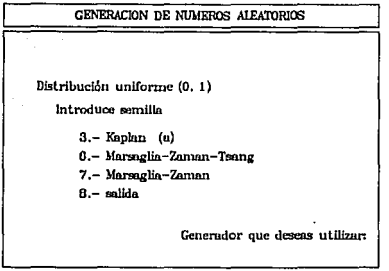
En caso de generarse números con distribución de probabilidad no uniforme se presenta a continuación la pantalla No.10.

Pantalla No.3



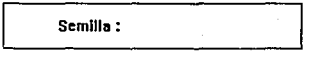
Algunos algoritmos nos permiten proporcionar la semilla del generador, esto nos da la facilidad de variar nuestra secuencia de datos alimentando el generador con una semilla diferente en cada corrida del programa. Si deseamos introducir semilla nuestra siguiente pantalla será la número 4, en caso contrario tendremos la número 9.

Pantalla No.4



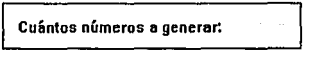
Estos tres algoritmos pueden ser inicializados con una semilla seleccionada por el usuario, recordando que sólo se modifica la secuencia generada, sin alterar el periodo de ésta.

Pantalla No.5



La condición con la que debe cumplir esta semilla es la de ser un número entero e impar.

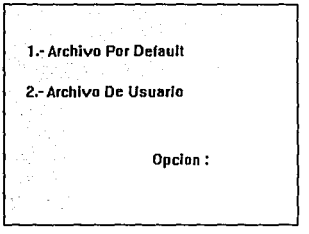
Pantalla No.6



Por último se pregunta al usuario la cantidad de números aleatorios que desea generar.

Si elegimos almacenarlos en archivo se presenta la siguiente pantalla, si no ha sido así, inicia el proceso de generación y la secuencia va apareciendo en la pantalla.

Pantalla No. 7



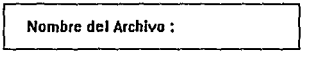
Si en la pantalla número 2 elegimos almacenar los datos en un archivo en este momento se solicita el nombre del archivo. El nombre de archivo por omisión esta dado por:

RNddmmaa

donde dd es el día del mes, mm es el número de mes del año y aa son los dos últimos dígitos del año en curso, ejemplo: RN311094.RAN. Todos los archivos se crean con extensión ".RAN".

Este programa esta en lenguaje Pascal y los archivos creados son de tipo texto.

Pantalla No.8

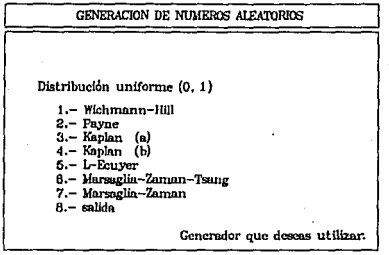


El archivo se crea en el mismo directorio donde se encuentre el programa, y en el nombre sólo se admiten caracteres alfanuméricos.

Una vez que se da el nombre del archivo se generan los números aleatorios apareciendo en pantalla la secuencia que se va generando. En seguida se muestra el diagrama de frecuencias (pantalla número 11).

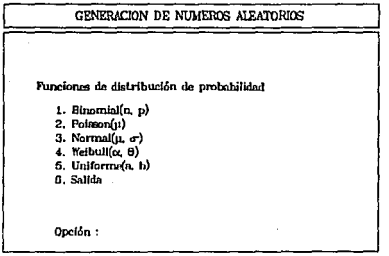
Si se están recuperando números de archivo, el programa indica cuantos datos a encontrado en el archivo y después va mostrando la secuencia en la pantalla.

Pantalla No.9



Estos son los siete algoritmos que generan números aleatorios con distribución uniforme en el intervalo (0,1)

Pantalla No.10



Se dispone de estas cinco funciones de distribución, de probabilidad

Una vez que seleccionamos alguna de ellas, se procede a introducir los valores de los parámetros correspondientes y aparece la pantalla número 6.

Pantalla No.11

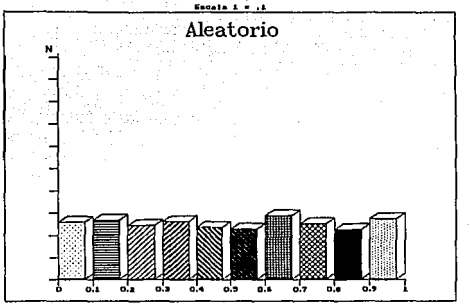


Diagrama de frecuencias de los números generados por el programa, el eje horizontal muestra los intervalos y el eje vertical las frecuencias.

Este diagrama sólo se genera si la secuencia se obtuvo con cualquiera de los siete generadores de números aleatorios con distribución uniforme.

SIMBOLOS USADOS

SIMBOLOS

$\cup$	unión de conjuntos
$\cap$	intersección de conjuntos
ϕ	conjunto vacío
$>$	mayor
$\geq$	mayor igual
$<$	menor
$\leq$	menor igual
$\neq$	diferente
$\int$	integración
Σ	sumatoria
∞	valor infinito
$\star$	multiplicación
$\chi \bmod \psi$	módulo: residuo de la división entera de χ y ψ
ν	grados de libertad
σ	desviación estandar
$\oplus$	or exclusivo
$\approx$	aproximadamente
e^{χ}	exponencial de χ
$\lfloor \chi \rfloor$	mayor entero menor o igual que χ (el "piso" de χ)
$\lceil \chi \rceil$	menor entero mayor o igual que χ (el "techo" de χ)
$\chi!$	χ factorial
$\lvert \chi \rvert$	valor absoluto de χ
$\mathrm{mcd}(h,k)$	máximo común divisor de h y k
Π	= 3.1415926535897932385
$\mathrm{int}(\chi)$	valor entero de χ
$\ln \chi$	logaritmo natural de χ
$\log \chi$	logaritmo base 10 de χ
$\blacksquare$	fin de una demostración

BIBLIOGRAFIA

Aho Alfred V., Hopcroft John E. & Ullman Jeffrey D., Data Structure and Algoritm, U.S.A., Addison Wesley Publishing Company, 1983.

Brian Wichmann and David Hill, "Building a Random-Number Generator", Byte, (march 1987) 127-128.

Canavos George C. Probabilidad y estadística - Aplicaciones y métodos, México, McGraw-Hill, 1986.

Goodman Seymour E. & Hedetniemi S.T., Introduction to the Design and Analysis of Algoritms, U.S.A., Ed. McGraw-Hill, 1977.

James, F. "A review of Pseudorandom Number Generators", Computer Physics Communications 60 (1990) 329-344.

Joyanes Aguilar Luis, Metodología de la programación, México, McGraw-Hill, 1988.

Kaplan, Howard L. "Effective Ramdom Seeding of Random Number Generators", Behavior Research Methods & Instrumentation, vol. XIII (2), (1981) 283-289.

Knuth, Donald E., El arte de programar ordenadores, vol. I : Algoritmos fundamentales, España, Ed. Reverté, 1980.

Knuth, Donald E., The Art of Computer Programming, vol. II: Seminumerical Algorithms, Addison-Wesley Publishing Co., Inc., Reading, Mass., 1969.

L'Ecuyer, Pierre "Efficient and Portable Combined Random Number Generators", Communications of the ACM, vol. XXX1, núm. 6, (June 1988) 742-774.

Lewis,T. G. and Payne,W. H., "Generalized Feedback Shift Register Pseudoram Number Algorithm". Journal of the Asociation for Computing Machinery, vol. XX, núm.3, (July 1973) 456-468.

Naylor Thomas H., et. al., Técnicas de simulación en computadora, México, Ed. Limusa, 1971.

Tremblay Jean-Paul y Bunt Richard B., Introducción a la ciencia de las computadoras: enfoque algorítmico, México, Ed. McGraw-Hill, 1982.

Wilde Daniel U., An Introduction to Computing Problem-Solving, Algoritms, and Data Structures, Englewood Cliffs, New Jersey,Ed. Prentice Hall, 1973.