su partida, siempre que haya uno disponible. Suponga que la probabilidad de que esté lloviendo en el momento de cualquier salida es$pags$. DejarX_{norte}indique el número de sombrillas disponibles en el lugar donde Ella llega después del número de caminata$norte$;$norte$=1,2, . . ., incluido el que posiblemente traiga consigo. Calcula la probabilidad límite de que llueva y no haya paraguas disponible.

1.31 Se suelta un ratón en el laberinto de la figura 1.10. De cada compartimento, el ratón elige uno de los compartimentos adyacentes con igual probabilidad, independientemente del pasado. El ratón pasa una cantidad de tiempo distribuida exponencialmente en cada compartimento. El tiempo medio de permanencia en cada uno de los compartimentos 1, 3 y 4 es de dos segundos; el tiempo medio de permanencia en los compartimentos 2, 5 y 6 es de cuatro segundos. Dejar$\{X_t$, t-0$\}$sea el proceso de salto de Markov que describe la posición del ratón por veces$t$-0. Suponga que el mouse comienza en el compartimento 1 en el momento$t$=0.

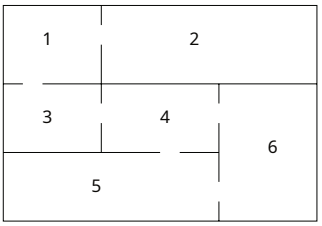

Figura 1.10: Un laberinto.

¿Cuáles son las probabilidades de que el ratón se encuentre en cada uno de los compartimentos 1,2, . . . ,6 en algún momento$t$muy lejos en el futuro?

1.32en un$M/M/\infty$-sistema de colas, los clientes llegan de acuerdo con un proceso de Poisson con tasa$a$. Cada cliente que ingresa es inmediatamente atendido por uno de una infinidad de servidores; por lo tanto, no hay cola. Los tiempos de servicio están distribuidos exponencialmente, con media 1/b. Todos los tiempos de servicio y entre llegadas son independientes. Dejar$X_t$sea el número de clientes en el sistema en ese momento$t$. Demuestre que la distribución límite deX_t, como$t\to\infty$, es Poisson con parámetroun/b.

Mejoramiento

1.33 Dejar**a**y deja**X**ser$norte$-vectores columna dimensionales. Muestra esa$\nabla_{\mathbf{x}}$**hacha**=**a**.

1.34 Dejar$A$ser un simétrico$norte\times norte$matriz y**X**frijol$norte$-vector columna dimensional. Muestra es$\frac{1}{2}$**X**$\nabla_{\mathbf{x}}$**X**=A**X**. ¿Cuál es el gradiente si$A$no es simetrico?

1.35 Demuestre que la distribución óptima**pags**•en el ejemplo 1.16 está dada por la distribución uniforme.

1.36 Derive el programa (1.71).

1.37 Considere el programa MinxEnt

$$\min_{\mathbf{pags}} \sum_{i=1}^{norte} pags_i \ln \frac{pags_i}{q_i}$$

$$\text{sujeto a: } \mathbf{pags} \geq \mathbf{0}, \quad A\mathbf{pags} = \mathbf{b}, \quad \sum_{i=1}^{norte} pags_i = 1,$$

dónde **pags** y **q** son vectores de distribución de probabilidad y A es un *metro* × *norte* matriz.

a) Demuestre que el Lagrangiano para este problema es de la forma

$$L(\mathbf{pags}, \boldsymbol{\lambda}, \beta, \boldsymbol{\mu}) = \mathbf{pags}^{\top}\ln(\mathbf{pags}) - \boldsymbol{\lambda}^{\top}(A\mathbf{pags} - \mathbf{b}) - \boldsymbol{\mu}\mathbf{pags} + \beta(\mathbf{1}^{\top}\mathbf{pag} - 1).$$

b) Muestra esa $pags_i = q_i \exp(-\beta - 1 + \mu_i + \sum_{j=1}^{metro} \lambda_j a_{ji})$, por $i = 1, \ldots, norte$.

C) Explique por qué, como resultado de las condiciones KKT, el óptimo $\boldsymbol{\mu}^*$ debe ser igual al vector cero.

d) Demuestre que la solución para este programa MinxEnt es exactamente la misma que para el programa donde se omiten las restricciones de no negatividad.

Otras lecturas

Una introducción fácil a la teoría de la probabilidad con muchos ejemplos es [13], y un libro de texto más detallado es [8]. Una referencia clásica es [6]. En [3] se da un tratamiento preciso y accesible de varios procesos estocásticos. Para la optimización convexa nos referimos a [2] y [7].

REFERENCIAS

1. ZI Botev, DP Kroese y T. Taimre. Métodos generalizados de entropía cruzada para la simulación y optimización de eventos raros. *Simulación: transacciones de la Society for Modeling and Simulation International*, 83(11):785–806, 2007.
2. S. Boyd y L. Vandenberghe. *Optimizacion convexa*. Prensa de la Universidad de Cambridge, Cambridge, Reino Unido, 2004.
3. E. Çinlar. *Introducción a los Procesos Estocásticos*. Prentice Hall, Englewood Cliffs, Nueva Jersey, 1975.
4. TM Cover y JA Thomas. *Elementos de la teoría de la información*. John Wiley & Sons, Nueva York, 1991.
5. CW Curtis. *Álgebra lineal: un enfoque introductorio*. Springer-Verlag, Nueva York, 1984.
6. W. Feller. *Una introducción a la teoría de la probabilidad y sus aplicaciones*, volumen 1. John Wiley & Sons, Nueva York, 2ª edición, 1970.
7. R. Fletcher. *Métodos prácticos de optimización*. John Wiley & Sons, Nueva York, 1987.
8. GR Grimmett y DR Stirzaker. *Procesos de probabilidad y aleatorios*. Oxford University Press, Oxford, 3.ª edición, 2001.
9. JN Kapur y HK Kesavan. *Principios de optimización de entropía con aplicaciones*. Prensa académica, Nueva York, 1992.

10. AI Khinchin. *Teoría de la información*. Publicaciones de Dover, Nueva York, 1957.

11. NV Krilov. *Introducción a la Teoría de Procesos Aleatorios*, volumen 43 de *Estudios de Posgrado en Matemáticas*. Sociedad Matemática Estadounidense, Providence, Rhode Island, 2002.

12. EL Lehmann. *Prueba de hipótesis estadísticas*. Springer-Verlag, Nueva York, 1997.

13. SM Ross. *Un primer curso de probabilidad*. Prentice Hall, Englewood Cliffs, Nueva Jersey, séptima edición, 2005.

CAPITULO 2

NÚMERO ALEATORIO, VARIABLE ALEATORIA Y GENERACIÓN DE PROCESOS ESTOCÁSTICOS

2.1 INTRODUCCIÓN

Este capítulo trata de la generación por computadora de números aleatorios, variables aleatorias y procesos estocásticos. En una simulación estocástica típica, la aleatoriedad se introduce en los modelos de simulación a través de variables aleatorias independientes distribuidas uniformemente. Estas variables aleatorias se utilizan luego como bloques de construcción para simular sistemas estocásticos más generales.

El resto de este capítulo está organizado de la siguiente manera. Comenzamos, en la Sección 2.2, con la generación de variables aleatorias uniformes. La sección 2.3 analiza los métodos generales para generar variables aleatorias unidimensionales. La sección 2.4 presenta algoritmos específicos para generar variables a partir de distribuciones continuas y discretas de uso común. En la Sección 2.5 analizamos la generación de vectores aleatorios. Las secciones 2.6 y 2.7 tratan la generación de procesos de Poisson, cadenas de Markov y procesos de salto de Markov. La generación de procesos Gaussianos y de difusión se da en las Secciones 2.8 y 2.9. Finalmente, la Sección 2.10 trata de la generación de permutaciones aleatorias.

2.2 GENERACIÓN DE NÚMEROS ALEATORIOS

En los primeros días de la simulación, la aleatoriedad se generaba mediante *manual* técnicas, como lanzar monedas, tirar dados, barajar cartas y girar la ruleta. Mas tarde,

dispositivos físicos, como diodos de ruido y contadores Geiger, se adjuntaron a las computadoras con el mismo propósito. La creencia predominante sostenía que solo los dispositivos mecánicos o electrónicos podían producir secuencias verdaderamente aleatorias. Aunque los dispositivos mecánicos todavía se usan ampliamente en juegos de azar y loterías, la comunidad de simulación por computadora abandonó estos métodos por varias razones: (1) los métodos mecánicos eran demasiado lentos para el uso general, (2) las secuencias generadas no podían reproducirse y (3) se encontró que los números generados exhiben tanto sesgo como dependencia. Aunque ciertos métodos modernos de generación física son rápidos y pasarían la mayoría de las pruebas estadísticas de aleatoriedad (p. ej., las que se basan en la radiación de fondo universal o el ruido de un chip de PC), su principal inconveniente sigue siendo la falta de repetibilidad. La mayoría de los generadores de números aleatorios actuales no se basan en dispositivos físicos sino en algoritmos simples que se pueden implementar fácilmente en una computadora. Son rápidos, requieren poco espacio de almacenamiento y pueden reproducir fácilmente una secuencia determinada de números aleatorios. Es importante destacar que un buen generador de números aleatorios captura todas las propiedades estadísticas importantes de las secuencias aleatorias verdaderas, aunque la secuencia se genera mediante un algoritmo determinista. Por esta razón, estos generadores a veces se denominan a pesar de que la secuencia es generada por un algoritmo determinista. Por esta razón, estos generadores a veces se denominan a pesar de que la secuencia es generada por un algoritmo determinista. Por esta razón, estos generadores a veces se denominan*pseudoaleatorio*.

La mayoría de los lenguajes de programación ya contienen un generador de números pseudoaleatorios incorporado. Por lo general, al usuario solo se le solicita que ingrese la semilla inicial,X_0, y al invocarlo, el generador de números aleatorios produce una secuencia de números independientes y uniformes (0,1) variables aleatorias. Por lo tanto, en este libro asumimos la disponibilidad de tal "caja negra" que sea capaz de producir un flujo de números pseudoaleatorios. En Matlab, por ejemplo, esto lo proporciona elrandfunción. La "semilla" del generador de números aleatorios, que se puede establecer mediante elanillofunción, determina qué flujo aleatorio se utiliza, y esto es muy útil para fines de prueba.

EJEMPLO 2.1Generación de variables aleatorias uniformes en Matlab

Este ejemplo ilustra el uso de larandfunción en Matlab para generar muestras a partir de latu(0,1) distribución. Para mayor claridad, hemos omitido el "respuesta = " salida en la sesión de Matlab a continuación.

```
> > rand                    % generar un número aleatorio uniforme
    0.0196
> > rand                    % generar otro número aleatorio uniforme
    0.823
> > rand(1,4)               % genera un vector aleatorio uniforme
    0,5252 0,2026 0,6721 0,8381
> > anillo(1234)            % establece la semilla en 1234
> > rand                    % generar un número aleatorio uniforme
    0.1915
> > anillo(1234)            % restablecer la semilla a 1234
> > rand
    0.1915                  % se repite el resultado anterior
```

Los métodos más simples para generar secuencias pseudoaleatorias utilizan los llamados *generadores lineales congruentes*, introducido en [12]. Estos generan una secuencia determinista de números por medio de la fórmula recursiva

$$X_{t+1} = hacha_t + C (\text{modificación } metro), \tag{2.1}$$

donde el valor inicial, X_0, se llama el *semilla* y el *una c*, y *metro* (todos los enteros positivos) se denominan *multiplicador*, la *incremento*, y el *módulo*, respectivamente. Nótese que aplicando el módulo *metro* operador en (2.1) significa que *hacha* + *C* se divide por *metro*, y el resto se toma como el valor de X_{t+1}. Así cada uno *estado* X_t sólo puede asumir un valor del conjunto $\{0,1, \ldots, m-1\}$, y las cantidades

$$tu_t = \frac{X_t}{metro} \tag{2.2}$$

llamó *números pseudoaleatorios*, constituyen aproximaciones a una secuencia verdadera de variables aleatorias uniformes. Nótese que la secuencia $X_0, X_1, X_2, \ldots$ se repetirá después de a lo sumo *metro* escalones y por tanto será periódica, con un plazo no superior a *metro*. Por ejemplo, deja $a = C = X_0 = 3$ y *metro* = 5. Entonces la sucesión obtenida de la fórmula recursiva $X_{t+1} = 3X_t + 3$ (módulo 5) es 3,2,4,0,3, que tiene un período 4. Una elección cuidadosa de *a*, *metro*, y *C* puede conducir a un generador lineal congruente que pasa la mayoría de las pruebas estadísticas estándar de uniformidad e independencia. Un ejemplo es el generador lineal congruente con *metro* $= 2^{31} - 1$, $a = 7^4$, y $C = 0$ por Lewis, Goodman y Miller [13]. Sin embargo, hoy en día, los generadores congruentes lineales ya no cumplen con los requisitos de las aplicaciones modernas de Monte Carlo (p. ej., [11]) y han sido reemplazados por algoritmos recursivos lineales más generales, que se analizan a continuación.

2.2.1 Generadores recursivos múltiples

A *generador recursivo múltiple* (MRG) de *ordenar* k está determinada por una secuencia de k Vectores de estado dimensional $\mathbf{X}_t = (X_{t-k+1}, \ldots, X_t)$, $t = 0,1,2, \ldots$, cuyos componentes satisfacen la recurrencia lineal

$$X_t = (a_1 X_{t-1} + \cdots + a_k X_{t-k}) \text{ modificación } m, \; t = k, k+1, \ldots \tag{2.3}$$

por algún módulo *metro*, multiplicadores $\{a_i, i = 1, \ldots, k\}$, y una semilla dada $\mathbf{X}_0 = (X_{-k+1}, \ldots, X_0)$. La duración máxima del período para este generador es $metro_k - 1$. Para producir algoritmos rápidos, todos menos algunos de los multiplicadores deben ser 0. Cuando *metro* es un entero grande, el flujo de salida de números aleatorios se obtiene a través de $tu_t = X_t / metro$.

Los MRG con períodos muy grandes se pueden implementar de manera eficiente al combinar varios MRG con períodos más pequeños, lo que produce *generadores recursivos múltiples combinados*.

EJEMPLO 2.2

Uno de los MRG combinados más exitosos es MRG32k3a por L'Ecuyer [9], que emplea dos MRG de orden 3, con recurrencias

$$X_t = (1403580 X_{t-2} - 810728 X_{t-3}) \text{ modificación } metro_1 \qquad (metro_1 = 2^{32} - 209),$$

$$Y_t = (527612 Y_{t-1} - 1370589 Y_{t-3}) \text{ modificación } metro_2 \qquad (metro_2 = 2^{32} - 22853),$$

y salida

$$tu_t = \begin{cases} \dfrac{X_t - Y_t + metro_1}{metro_1 + 1} & \text{si } X_t - Y_t \\ \dfrac{X_t - Y_t}{metro_1 + 1} & \text{si } X_t > Y_t \end{cases}$$

La duración del período es de aproximadamente 3×10^{57}. El generador MRG32k3a pasa todas las pruebas estadísticas en el traje de prueba más completo de la actualidad *PruebaU01* [11] y se ha implementado en muchos paquetes de software, incluidos Matlab, Mathematica, la biblioteca MKL de Intel, SAS, VSL, Arena y Automod. También es el generador central en el paquete de simulación SSJ de L'Ecuyer y es fácilmente ampliable para generar múltiples flujos aleatorios.

2.2.2 Generadores lineales de módulo 2

Los buenos generadores aleatorios deben tener espacios de estado muy grandes. Para un generador lineal congruente, esto significa que el módulo *metro* debe ser un entero grande. Sin embargo, para múltiples generadores recursivos, no es necesario tomar un módulo grande, ya que la duración del período puede ser tan grande como $metro_k - 1$. Debido a que las operaciones binarias son en general más rápidas que las operaciones de punto flotante (que a su vez son más rápidas que las operaciones con enteros), tiene sentido considerar MRG y otros generadores de números aleatorios que se basan en recurrencias lineales módulo 2. Un marco general para tales operaciones aleatorias generadores de números se da en [10], donde el estado es un k vector de bits $\mathbf{X}_t = (X_{yo,1}, \ldots, X_{t,k})$ que se asigna a través de una transformación lineal a un w vector de salida de bits $\mathbf{Y}_t = (Y_{yo,1}, \ldots, Y_{t,w})$, de donde el número aleatorio $tu_t \in (0,1)$ se obtiene por *diezmado bit a bit* como sigue:

Algoritmo 2.2.1: Generador Genérico Módulo 2 de Recurrencia Lineal

aporte : Distribución de semillas m en el espacio de estado $S = \{0,1\}^k$ y tamaño de la muestra *norte*.
producción: Secuencia $tu_1, \ldots, tu_{norte}$ de números pseudoaleatorios.

1 dibujar la semilla $\mathbf{X}_0$ de la distribución m. // inicializar
2 **por** $t = 1$ **a** *norte* **hacer**
3 $\quad \mathbf{X}_t \leftarrow A\mathbf{X}_{t-1}$ // transición
4 $\quad \mathbf{Y}_t \leftarrow B\mathbf{X}_t$ // transformación de salida
5 $\quad tu_t \leftarrow \sum_{w=1}^{t} Y_{yo,2-}$ // diezmar
6 **devolver** $tu_1, \ldots, tu_{norte}$

Aquí, A y B son $k \times k$ y $w \times k$ matrices binarias, respectivamente, y todas las operaciones se realizan módulo 2. En particular, la suma corresponde a la bit a bit XOR operación (en particular, 1 + 1 = 0). el entero w se puede considerar como la longitud de palabra de la computadora (es decir, $w = 32$ o 64). Por lo general (pero hay excepciones, consulte [10]) k se toma mucho más grande que w.

EJEMPLO 2.3 Tornado de Mersenne

Matsumoto y Nishimura [16] introdujeron un popular generador de módulo 2. La dimensión k del vector de estado $\mathbf{X}_t$ en el Algoritmo 2.2.1 es en este caso $k = wn$, dónde w es la longitud de la palabra (por defecto 32) y *norte* un entero grande (predeterminado 624). Se puede mostrar que la duración del período para la elección predeterminada de parámetros es de $2^{w(n-1)+1} - 1 = 2^{19937} - 1$. En lugar de tomar el estado $\mathbf{X}_t$ como un $wn \times 1$ vector, es conveniente considerarlo como un $norte \times w$ matriz con filas $\mathbf{X}_t, \ldots, \mathbf{X}_{t+n-1}$. A partir de las hileras de semillas $\mathbf{X}_0, \ldots, \mathbf{X}_{n-1}$, en cada paso $t = 0,1,2, \ldots$ la $(t + norte)$-ésima fila se calcula de acuerdo con las siguientes reglas:

1. Toma el primero r pedazos de $\mathbf{X}_t$ y el último $w-r$ pedazos de $\mathbf{X}_{t+1}$ y concentrarlos juntos en un vector binario $\mathbf{X}$.

2. Aplique la siguiente operación binaria a $\mathbf{X} = (X_1, \ldots, X_w)$ para dar un nuevo vector binario $\mathbf{X}$:

$$\mathbf{X} = \begin{cases} \mathbf{X} \quad 1 & \text{si } X_w = 0, \\ (\mathbf{X} \quad 1) \oplus \mathbf{a} & \text{si } X_w = 1. \end{cases}$$

3. Deja $\mathbf{X}_{t+norte} = \mathbf{X}_{t+metro} \oplus \mathbf{X}$.

Aquí $\oplus$ representa el XOR operación y 1 para la operación de desplazamiento a la derecha (cambie los bits una posición a la derecha, agregando un 1 desde la izquierda). El vector binario $\mathbf{a}$ y los numeros $metro$ y r son especificados por el usuario (ver abajo).

La salida en el paso t del algoritmo se realiza como la aniquilación bit a bit de un vector $\mathbf{y}$ que se obtiene a través de los siguientes cinco pasos:

1. $\mathbf{y} = \mathbf{X}_{t+norte}$
2. $\mathbf{y} = \mathbf{y} \oplus (\mathbf{y} \quad tu)$
3. $\mathbf{y} = \mathbf{y} \oplus ((\mathbf{y} \quad s) \ \& \mathbf{b})$
4. $\mathbf{y} = \mathbf{y} \oplus ((\mathbf{y} \quad v) \ \& \mathbf{C})$
5. $\mathbf{y} = \mathbf{y} \oplus (\mathbf{y} \quad yo)$

Aquí & denota el Y operación (multiplicación bit a bit), y ($\mathbf{y}$ s) indica que $\mathbf{y}$ hace un desplazamiento a la izquierda por s posiciones, agregando 0s desde la derecha; similar, ($\mathbf{y}$ tu) es un desplazamiento a la derecha por tu posiciones, sumando 1 desde la izquierda. los vectores $\mathbf{b}$ y $\mathbf{C}$ así como los números enteros tu y s son proporcionados por el usuario. Los parámetros recomendados para el algoritmo son

$$(w, n, m, r) = (32, 624, 397, 31)$$

y

$$(\mathbf{a}, \mathbf{b}, \mathbf{C}, tu, s, v, l) = (9908\text{B}0\text{DF}_{\text{dieciséis}}, 9\text{D}2\text{C}5680_{\text{dieciséis}}, \text{EFC}60000_{\text{dieciséis}}, 11, 7, 15, 18),$$

donde el subíndice $_{\text{dieciséis}}$ indica notación hexadecimal; por ejemplo, $7\text{B}_{\text{dieciséis}} = 0111\ 1101$.

Como ejemplo concreto del funcionamiento del tornado de Mersenne, suponga que los valores iniciales son como en la Tabla 2.1 (suponiendo parámetros predeterminados).

Tabla 2.1: Estado inicial del tornado de Mersenne.

$\mathbf{X}_0$	00110110110100001010111000101001
$\mathbf{X}_1$	10101010101010101010101010101011
$\mathbf{X}_2$	10001000100000001111100010101011
...	...
$\mathbf{x}_{metro}$	10010101110100010101011110100011
...	...
$\mathbf{X}_{n-1}$	00100111001010101110110110110010

Generemos el primer número aleatorio. Suponer que $r=31$, entonces,

$$\mathbf{X}=0\overbrace{01101101101000\ldots0}^{r\text{ pedazos de }\mathbf{X}_0}\overbrace{101011100010100}^{w-r\text{ pedazos de }\mathbf{X}_1}.$$

El bit menos significativo (el bit más a la derecha) de $\mathbf{X}$ es 1, entonces $\mathbf{X}$ se obtiene desplazando a la derecha $\mathbf{X}$ y XOR-ing el vector resultante con $\mathbf{a}$, asi que

$$\mathbf{X}=10011011011010000101011100010100\oplus 00000001011010100011010010111001$$
$$= 10011010000000100110001110101101.$$

Completamos el cálculo de $\mathbf{X}_{norte}=\mathbf{X}_{metro}\oplus\mathbf{X}$ y obten

$$\mathbf{X}_{norte}=10010101110100010101011110100011\oplus 10011010000000100110001110101101 =$$
$$= 00001111110100110011010000001110.$$

A continuación, determinamos el vector de salida $\mathbf{y}$ en cinco pasos.

1. $\mathbf{y}=\mathbf{X}_{norte}=00001111110100110011010000001110.$

2. $\mathbf{y}=\mathbf{y}\oplus(\mathbf{y}$ 11)
$= 00001111110100110011010000001110\oplus 11111111111000011111101001100110 =$
$11110000001100101100111001101000.$

3. $\mathbf{y}=\mathbf{y}\oplus((\mathbf{y}$ 7) y $9D2C5680_{dieciséis}$)

$= 11110000001100101100111001101000$
$\oplus (00011001011001110011010000000000\&00000000001100101100111001101000) =$
$11110000001100101100111001101000\oplus 00000000001000100000010000000000 =$
$11110000000100001100101001101000.$

4. $\mathbf{y}=\mathbf{y}\oplus((\mathbf{y}$ 15) & 00000000000000000110001111110111) — similar a 3, lo que resulta en

$$\mathbf{y}=11110001010100001111110100.$$

5. $\mathbf{y}=\mathbf{y}\oplus(\mathbf{y}$ 18) — similar a 2, lo que da como resultado

$$\mathbf{y}=00001110101011110000011000111100.$$

Teniendo en cuenta que la representación decimal de final $\mathbf{y}$ es igual a $1.0130\cdot 10^9$, el algoritmo devuelve $1.0130\cdot 10^9/2^{32}-1 = 0.2359$ como salida.

Para una versión actualizada del código del Mersenne twister nos referimos a

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

Tenga en cuenta que para la inicialización, un completo $w\times norte$ la matriz tiene que ser especificada. Esto es menudo se hace ejecutando un generador lineal básico. El algoritmo es muy rápido cuando se implementa en lenguajes compilados como C y pasa la mayoría de las pruebas estadísticas, pero se sabe que se recupera muy lentamente de los estados cercanos a cero; véase [11, página 23].

2.3 GENERACIÓN DE VARIABLES ALEATORIAS

En esta sección analizamos varios métodos generales para generar variables aleatorias unidimensionales a partir de una distribución prescrita. Consideramos el método de transformada inversa, el método de alias, el método de composición y el método de aceptación-rechazo.

2.3.1 Método de transformada inversa

Dejar X ser una variable aleatoria con cdf F. Ya que F es una función no decreciente, la función inversa F^{-1} se puede definir como

$$F^{-1}(y) = \text{INF}\{X : F(X) \geq y\}, \quad 0 \leq y \leq 1. \tag{2.4}$$

(Los lectores que no estén familiarizados con la noción inf deberían leer min.) Es fácil demostrar que si $tu \sim$ tu(0,1), entonces

$$X = F^{-1}(tu) \tag{2.5}$$

tiene CDF F. Es decir, desde F es invertible y PAGS($tu \leq tu$) = tu, tenemos

$$\text{PAGS}(X \leq X) = \text{PAGS}(F^{-1}(tu) \leq X) = \text{PAGS}(tu \leq F(X)) = F(X). \tag{2.6}$$

Así, para generar una variable aleatoria X con CDF *efectos especiales* $= F^{-1}(tu)$. La Figura 2.1 ilustra el siguiente algoritmo de tra inversa:

Algoritmo 2.3.1: Método de transformada inversa

aporte : Función de distribución acumulativa F. **producción:** Variable aleatoria X distribuido de acuerdo con

1 Generar tu detu(0,1).
2 $X \leftarrow F^{-1}(tu)$
3 **devolver** X

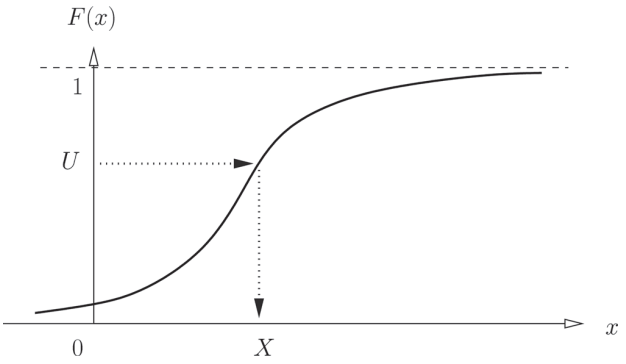

Figura 2.1: Método de transformada inversa.

EJEMPLO 2.4

Generar una variable aleatoria a partir del pdf.

$$F(X) = \begin{cases} 2X, & 0 \le X \le 1, \\ 0 & \text{de lo contrario.} \end{cases} \tag{2.7}$$

el fdf es

$$F(X) = \begin{cases} 0, & X < 0, \\ \int_0^X 2y\,dy = X^2, & 0 \le X \le 1, \\ 1, & X > 1. \end{cases}$$

Aplicando (2.5), tenemos

$$X = F^{-1}(tu) = \sqrt{tu}$$

Por lo tanto, para generar una variable aleatoria X del pdf (2.7), primero genera una variable aleatoria tu detu(0,1) y luego sacar su raíz cuadrada.

EJEMPLO 2.5 Estadísticas de pedidos

Dejar $X_1, \ldots, X_{norte}$ ser iid variables aleatorias con cdf F. Deseamos generar variables aleatorias $X_{(norte)}$ y $X_{(1)}$ que se distribuyen según las estadísticas de pedido max($X_1, \ldots, X_{norte}$) y min($X_1, \ldots, X_{norte}$), respectivamente. Del Ejemplo 1.7 vemos que la función de distribución acumulada de $X_{(norte)}$ y $X_{(1)}$ son $F_{norte}(X) = [F(X)]^{norte}$ y $F_1(X) = 1 - [1 - F(X)]^{norte}$, respectivamente. Aplicando (2.5), obtenemos

$$X_{(norte)} = F^{-1}(tu^{1/norte}),$$

y, desde el $1 - tu$ también es detu(0,1),

$$X_{(1)} = F^{-1}(1 - tu^{1/norte}).$$

En el caso especial en que $F(X) = X$, eso es, $X_i \sim$ tu(0,1), tenemos

$$X_{(norte)} = tu^{1/norte} \quad \text{y} \quad X_{(1)} = 1 - tu^{1/norte}.$$

EJEMPLO 2.6 Dibujar a partir de una distribución discreta

Dejar $X \sum$ Sea una variable aleatoria discreta con $PAGS(X = X_i) = pags_i$, $i = 1,2, \ldots,$ w $\sum$ yo $_i pags_i = 1$ y $X_1 < X_2 < \ldots$ el cdf F de X es dado por $F(X) = \sum_{i: X_i \le X} pags_i$, $i = 1,2, \ldots$ y se ilustra en la Figura 2.2.

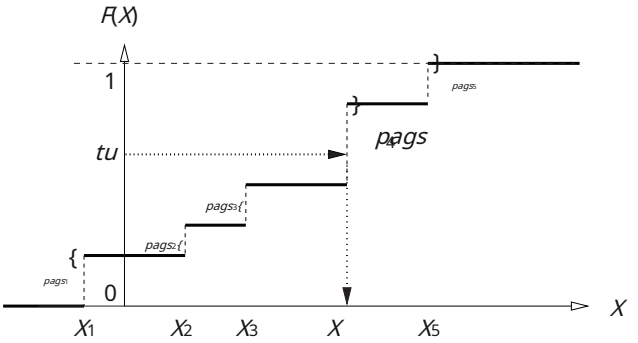

Figura 2.2: Método de transformada inversa para una variable aleatoria discreta.

Por lo tanto, el algoritmo para generar una variable aleatoria a partir de F se puede escribir de la siguiente manera:

Algoritmo 2.3.2: Método de transformada inversa para una distribución discreta

aporte : Función de distribución acumulada discreta F. **producción:** Variable aleatoria discreta X distribuido de acuerdo a F.

1 Generar $tu \sim$ tu(0,1).

2 Encuentre el entero positivo más pequeño, k, tal que $tu \le F(X_k)$. Dejar $X \leftarrow X_k$.

3 **devolver** X

Gran parte del tiempo de ejecución en el Algoritmo 2.3.2 se dedica a hacer las comparaciones del Paso 2. Este tiempo se puede reducir usando técnicas de búsqueda eficientes (ver [2]).

En general, el método de la transformada inversa requiere que la cdf subyacente, F, existen en una forma para la cual la función inversa correspondiente F^{-1} se puede encontrar analíticamente o algorítmicamente. Las distribuciones aplicables son, por ejemplo, las distribuciones exponencial, uniforme, Weibull, logística y Cauchy. Desafortunadamente, para muchas otras distribuciones de probabilidad, es imposible o difícil encontrar la transformada inversa, es decir, resolver

$$F(X) = \int_{-\infty}^{X} F(t)\, \mathrm{d}t = tu$$

con respecto a X. Incluso en el caso de que F^{-1} existe de forma explícita, el método de transformada inversa puede no ser necesariamente el método de generación de variables aleatorias más eficiente (ver [2]).

2.3.2 Método de alias

Una alternativa al método de transformada inversa para generar variables aleatorias discretas, que no requiere técnicas de búsqueda que consumen mucho tiempo según el Paso 2 del Algoritmo 2.3.2, es el llamado *método de alias* [19]. Se basa en el hecho de que un discreto arbitrario $norte$-punto pdf F, con

$$F(X_i) = \text{PAGS}(X = X_i),\ i = 1, \ldots, norte,$$

puede representarse como una mezcla igualmente ponderada de $norte$ pdf, $q_{(k)}$, $k = 1, \ldots, norte$, teniendo cada uno como máximo dos componentes distintos de cero. es decir, cualquier $norte$-punto pdf F puede ser

representado como

$$F(X) = \frac{1}{norte} \sum_{k=1}^{norte} q_{(k)}(X) \tag{2.8}$$

para archivos PDF de dos puntos adecuadamente definidos $q_{(k)}$, $k=1, \ldots, norte$; ver [19].

El método de alias es bastante general y eficiente, pero requiere una configuración inicial y almacenamiento adicional para el $norte$ pdf, $q_{(k)}$. En [2] se puede encontrar un procedimiento para calcular estas pdf de dos puntos. Una vez establecida la representación (2.8), la generación a partir de F es simple y se puede escribir de la siguiente manera:

Algoritmo 2.3.3: Método de alias

aporte : PDF de dos puntos $q_{(k)}$, $k=1, \ldots, norte$ representando pdf discreto F.
producción: Variable aleatoria discreta X distribuido de acuerdo a F.
1 Generar $tu \sim tu(0,1)$ y establecer $k \leftarrow nU$.
2 Generar X del pdf de dos puntos $q_{(k)}$.
3 **devolver** X

2.3.3 Método de composición

Este método asume que un cdf, F, se puede expresar como un *mezcla* de cdfs $\{GRAMO_i\}$, eso es,

$$F(X) = \sum_{i=1}^{metro} pags_i GRAMO(X), \tag{2.9}$$

dónde

$$pags_i > 0, \quad \sum_{i=1}^{metro} pags_i = 1.$$

Dejar $X_i \sim GRAMO_i$ y deja Y Sea una variable aleatoria discreta con PAGS$(Y=i) = pags_i$, independiente de X_i, para 1 - i - $metro$. Entonces una variable aleatoria X con CDF F se puede representar como

$$X = \sum_{i=1}^{metro} X_i yo_{\{Y=i\}}.$$

De ello se deduce que para generar X de F, primero debemos generar la variable aleatoria discreta Y y luego, dado $Y=i$, generar X_i de $GRAMO_i$. Así tenemos el siguiente método:

Algoritmo 2.3.4: Método de composición

aporte : CDF de mezcla F.
producción: Variable aleatoria X distribuido de acuerdo a F.
1 Generar la variable aleatoria Y de acuerdo a PAGS$(Y=i) = pags_i$, $i=1, \ldots, m$.
2 Dado $Y=i$, generar X de la CDF $GRAMO_i$.
3 **devolver** X

2.3.4 Método de aceptación-rechazo

Los métodos de transformación inversa y de composición tratan directamente con la función de distribución acumulativa del método de aceptación-rechazo aleatorio v, es un método indirecto de von Neumann. Se puede aplicar cuando los anteriores fallan o resultan ser computacionalmente ineficientes.

Para introducir la idea, supongamos que el *objetivo* pdf *F* (el pdf del que queremos muestrear) está limitado por un intervalo finito [*un, b*] y es cero fuera de este intervalo (ver Figura 2.3). Dejar

$$C = \text{sorber}\{F(X) : X \in [un, b]\}.$$

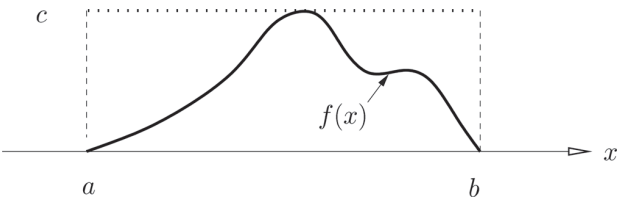

Figura 2.3: El método de aceptación-rechazo.

En este caso, generar una variable aleatoria $Z \sim F$ es sencillo, y se puede hacer usando los siguientes pasos de aceptación-rechazo:

1. Generar $X \sim \text{tu}(un, b)$.
2. Generar $Y \sim \text{tu}(0, C)$ independientemente de X.
3. Si $Y < F(X)$, devolver $Z = X$. De lo contrario, regrese al Paso 1.

Es importante notar que cada vector generado (X, Y) se distribuye uniformemente sobre el rectángulo $[un, b] \times [0, C]$. Por lo tanto el par aceptado (X, Y) se distribuye uniformemente bajo la gráfica F. Esto implica que la distribución de los valores aceptados de X tiene el pdf deseado F.

Podemos generalizar esto de la siguiente manera: Sea *gramo* ser una densidad arbitraria tal que $\varphi(X) = C\ gramo(X)$ *mayoriza* $F(X)$ para alguna constante C (figura 2.4); eso es, $\varphi(X) - F(X)$ para todos X. Tenga en cuenta que por necesidad $C \geq 1$. Llamamos *gramo*(X) la *propuesta* pdf y suponga que es fácil generar variables aleatorias a partir de él.

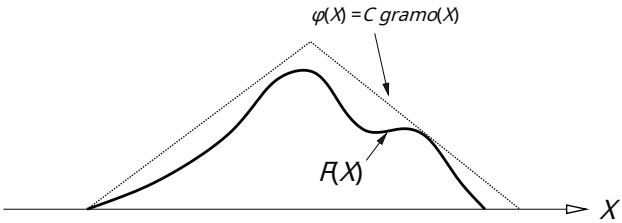

Figura 2.4: El método de aceptación-rechazo con una función mayorizadora φ.

El algoritmo de aceptación-rechazo se puede escribir de la siguiente manera:

Algoritmo 2.3.5: Método de aceptación-rechazo

aporte : pdf *gramo* y constante *C* tal que $CG(X) \geq F(X)$ para todos X.
producción: Variable aleatoria X distribuido segun pdf F.

```
1 fundar ← falso
2 mientras no fundar hacer
3     Generar X de gramo(X). Generar Y ~ tu(0,
4     C gramo(X)). si Y ≤ F(X) después fundar ←
5     verdadero
6 devolver X
```

La base teórica del método de aceptación-rechazo la proporciona el siguiente teorema:

Teorema 2.3.1 *La variable aleatoria generada según el Algoritmo 2.3.5 tiene la pdf deseada $F(X)$.*

Prueba: Defina los siguientes dos subconjuntos:

$$A = \{(x, y) : 0 \leq y \leq CG(X)\} \text{ y } B = \{(x, y) : 0 \leq y \leq F(X)\}, \tag{2.10}$$

que representan las áreas debajo de las curvas $CG(X)$ y $F(X)$, respectivamente. primero observe que las Líneas 3 y 4 del Algoritmo 2.3.5 implican que el vector aleatorio (X, Y) se distribuye uniformemente en A. Para ver esto, deja $q(x, y)$ denote el pdf conjunto de (X, Y), y deja $q(y \mid x)$ denota la función de densidad de probabilidad condicional de Y dado $X = X$. Entonces tenemos

$$q(x, y) = \begin{cases} gramo(X) q(y \mid x) & \text{si } (x, y) \in A, \text{ de lo} \\ 0 & \text{contrario.} \end{cases} \tag{2.11}$$

Ahora la línea 4 dice que $q(y \mid x)$ es igual a $1 / (CG(X))$ por $y \in [0, CG(X)]$ y es cero en caso contrario. Por lo tanto, $q(x, y) = C^{-1}$ para cada $(x, y) \in A$.

Dejar (X_*, Y_*) ser el primer punto aceptado, es decir, el primer punto que está en B. Dado que el vector (X, Y) se distribuye uniformemente en A, el vector (X_*, Y_*) se distribuye uniformemente en B. Además, dado que la zona de B es igual a la unidad, la pdf conjunta de (X_*, Y_*) en B igual a la unidad. Por lo tanto, la función de densidad de probabilidad marginal de $Z = X_*$ es

$$\int_0^{F(X)} 1 \text{ día} y = F(X).$$

los *eficiencia* del Algoritmo 2.3.5 se define como

$$\text{PAGS}((X, Y) \text{ se acepta}) = \frac{\text{área } B}{\text{área } A} = \frac{1}{C}. \tag{2.12}$$

A menudo, se utiliza una versión ligeramente modificada del Algoritmo 2.3.5. Esto es porque $Y \sim \text{tu}(0, C\ gramo(X))$ en la línea 4 es lo mismo que configurar $Y = CU\ g(X)$, dónde $tu \sim \text{tu}(0,1)$, y entonces podemos escribir $Y \leq F(X)$ en la línea 5 como $tu \leq F(X) / (C\ gramo(X))$. En otras palabras, generar X de $gramo(X)$ y aceptarlo con probabilidad $F(X) / (C\ gramo(X))$; de lo contrario rechazar X e intenta de nuevo. Por lo tanto, la versión modificada del Algoritmo 2.3.5 se puede reescribir de la siguiente manera:

Algoritmo 2.3.6: Método de aceptación-rechazo modificado

aporte : pdf *gramo* y constante *C* tal que $CG(X) \geq F(X)$ para todos *X*.
producción: Variable aleatoria *X* distribuido segun pdf *F*.

```
fundar ← falso
mientras no fundar hacer
    Generar X de gramo(X).
    Generar tu detu(0,1) independientemente de X. si tu
    ≤ F(X)/(CG(X)) después fundar ← verdadero
devolver X
```

EJEMPLO 2.7 Ejemplo 2.4 (Continuación)

Mostramos cómo generar una variable aleatoria *Z* del pdf

$$F(X) = \begin{cases} 2X, & 0 < X < 1, \\ 0 & \text{de lo contrario,} \end{cases}$$

utilizando el método de aceptación-rechazo. Para simplificar, tome $gramo(X) = 1, 0 < X < 1$, y $C = 2$. En este caso nuestra propuesta de distribución es simplemente la distribución uniforme en (0,1). Como consecuencia, $F(X)/(C\ gramo(X)) = X$ y el Algoritmo 2.3.6 se convierte en: seguir generando *X* y *tu* independientemente detu(0,1) hasta *tu* ≤ *X*; luego regresa *X*. Tenga en cuenta que este ejemplo es meramente ilustrativo, ya que es más eficiente simular a partir de este pdf utilizando el método de transformación inversa.

Como consecuencia de (2.12), la eficiencia del método de aceptación-rechazo modificado está determinada por la probabilidad de aceptación *pags* = PAGS(*tu* ≤ $F(X)/(CG(X))$) = PAGS($Y \leq F(X)$) = $1/C$ para cada ensayo (*X, tu*). Dado que los ensayos son independientes, el número de ensayos, *norte*, ante un par exitoso (*Z, U*) ocurre tiene la siguiente distribución geométrica:

$$\text{PAGS}(norte = norte) = pags(1 - pags)^{n-1},\ norte = 1, 2, \ldots, \tag{2.13}$$

con el número esperado de intentos igual a $1/pags = C$.

Para que este método sea de interés práctico, se deben utilizar los siguientes criterios al seleccionar la densidad propuesta *gramo*(*X*):

1. Debería ser fácil generar una variable aleatoria a partir de *gramo*(*X*).
2. La eficiencia, $1/C$, del procedimiento debe ser grande; eso es, *C* debe estar cerca de 1 (que ocurre cuando *gramo*(*X*) esta cerca de *F*(*X*)).

EJEMPLO 2.8

Generar una variable aleatoria *Z* de la densidad semicircular

$$F(X) = \frac{2\sqrt{\ }}{\pi R^2} \overline{R^2 - X^2},\ -R \leq X \leq r$$

Considere que la distribución de la propuesta es uniforme en $[-R, R]$; es decir, tomar *gramo* $(X) = 1/(2R),\ -R \leq X \leq R$ y elige *C* lo más pequeño posible de tal manera que $CG(X) \geq$

$f(x)$; por eso $c = 4/\pi$. Entonces el Algoritmo 2.3.6 conduce al siguiente algoritmo de generación:

1. Generar dos variables aleatorias independientes, u_1 y u_2, de U(0,1).

2. Usar u_2 para generar X de $g(x)$ a través del método de la transformada inversa, a saber $X = (2u_2 - 1)$, y calcular

$$\frac{f(x)}{c\,g(x)} = \sqrt{1 - (2u_2 - 1)^2}.$$

3. Si $u_1 \le f(x)/(c\,g(x))$, que es equivalente a $(2u_2 - 1)^2 \le 1 - u_2$, devolver $Z = X = (2u_2 - 1)$; de lo contrario, regrese al Paso 1.

El número esperado de intentos para este algoritmo es $c = 4/\pi$, y la eficiencia es $1/c = \pi/4 \approx 0.785$.

2.4 GENERACIÓN A PARTIR DE DISTRIBUCIONES DE USO COMÚN

Las siguientes dos subsecciones presentan algoritmos para generar variables a partir de distribuciones continuas y discretas de uso común. De los numerosos algoritmos disponibles (por ejemplo, [2]), hemos tratado de seleccionar aquellos que son razonablemente eficientes y relativamente simples de implementar.

2.4.1 Generación de variables aleatorias continuas

2.4.1.1 Distribución exponencial Empezamos aplicando el método de la transformada inversa a la distribución exponencial. Si $X \sim \text{Exp}(\lambda)$, entonces su cdf F es dado por

$$F(x) = 1 - e^{-\lambda x}, \qquad x \ge 0. \tag{2.14}$$

Por lo tanto, resolver $u = F(x)$ en términos de x da

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

Porque $u \sim U(0,1)$ implica $1 - u \sim U(0,1)$, llegamos al siguiente algoritmo:

Algoritmo 2.4.1: Generación de una variable aleatoria exponencial

aporte: $\lambda > 0$

producción: Variable aleatoria X distribuido de acuerdo a Exp(λ).

1 Generar $u \sim U(0,1)$.

2 $X \leftarrow -\ln u/\lambda$

3 **devolver** X

Existen muchos procedimientos alternativos para generar variables a partir de la distribución exponencial. Se remite al lector interesado a [2].

2.4.1.2 Distribución normal (gaussiana) Si $X \sim \text{NORTE}(\mu, \sigma^2)$, su pdf está dada por

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x-m)^2}{2\sigma^2}\right\}, \qquad -\infty < x < \infty, \tag{2.15}$$

dónde m es la media (o expectativa) y σ^2 la varianza de la distribución.

Dado que la inversión de la cdf normal es numéricamente ineficiente, el método de la transformada inversa no es muy adecuado para generar variables aleatorias normales, por lo que se deben idear otros procedimientos. Consideramos solo la generación de NORTE(0,1) (variables normales estándar), ya que cualquier $Z \sim \text{NORTE}(\mu, \sigma^2)$ se puede representar como $Z = m + \sigma X$, dónde X es de NORTE(0,1). Uno de los primeros métodos para generar variables a partir de NORTE(0,1) fue desarrollado por Box y Muller de la siguiente manera:

Dejar X y Y ser dos variables aleatorias normales estándar independientes; Y (X, Y) es un punto aleatorio en el plano. Dejar (R, Θ) sean las coordenadas polares correspondientes. el pdf conjunto $f_{R,\Theta}$ de R y Θ está dado por

$$f_{R,\Theta}(r, \theta) = \frac{1}{2\pi} \text{mi}^{-r^2/2} r \qquad \text{por } r > 0 \text{ y } \theta \in [0, 2\pi).$$

Esto se puede ver escribiendo X y y en términos de r y θ, Llegar

$$x = r \text{ porque } \theta \text{ y } y = r \text{ pecado } \theta \tag{2.16}$$

El jacobiano de esta transformación de coordenadas es

$$\det\begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{pmatrix} = \begin{vmatrix} \text{porque } \theta & -r \text{ pecado } \theta \\ \text{pecado } \theta & r \text{ porque } \theta \end{vmatrix} = r$$

El resultado ahora se sigue de la regla de transformación (1.20), observando que la articulación pdf de X y Y es $f_{X,Y}(x, y) = \frac{1}{2\pi} \text{mi}^{-(x^2+y^2)/2}$. No es difícil comprobar que R y Θ son independientes, que $\Theta \sim \text{tu}[0, 2\pi)$, a √ y eso $\text{PAGS}(R > r) = \text{mi}^{-r^2/2}$. este yo √ y eso R tiene la misma distribución que $\sqrt{V}$, con $V \sim \text{Exp}(1/2)$. A saber, $\text{PAGS}(V > v) = \text{PAGS}(V > v^2) = \text{mi}^{-v^2/2}$, $v > 0$. Así, tanto Θ como R son fáciles de generar y se transforman mediante (2.16) en variables aleatorias normales estándar independientes. Esto conduce al siguiente algoritmo:

Algoritmo 2.4.2: Generación de variables aleatorias normales: enfoque de Box-Muller

producción: Variables aleatorias normales estándar independientes X y Y.

1 Generar dos variables aleatorias independientes, tu_1 y tu_2, detu(0,1).
2 $X \leftarrow (-2 \text{ en } tu_1)^{1/2} \text{ porque } (2\pi U_2)$
3 $Y \leftarrow (-2 \text{ en } tu_1^{1/2}\ _1) \text{ pecado}(2\pi U_2)$
4 **devolver** X, Y

Un método alternativo de generación de NORTE(0,1) se basa en el método de aceptación-rechazo. Primero, tenga en cuenta que para generar una variable aleatoria Y de NORTE(0,1), primero se puede generar una variable aleatoria positiva X del pdf

$$f(x) = \sqrt{\frac{2}{\pi}}\, \text{mi}^{-x^2/2}, \qquad x > 0, \tag{2.17}$$

y luego asignar a X un signo aleatorio. La validez de este procedimiento se deriva de la simetría de la distribución normal estándar alrededor de cero.

Para generar una variable aleatoria X de (2.17), enlazamos $f(x)$ por $C \cdot g(x)$, dónde $g(x) = \mathrm{mi}^{-x}$ es el pdf de la Exp(1) distribución. La constante más pequeña C tal que $f(x) \le Cg(x)$ es $\sqrt{2e/\pi}$ (ver Figura 2.5). Por lo tanto, la eficiencia de este método es $\sqrt{\pi/2e} \approx 0.76$.

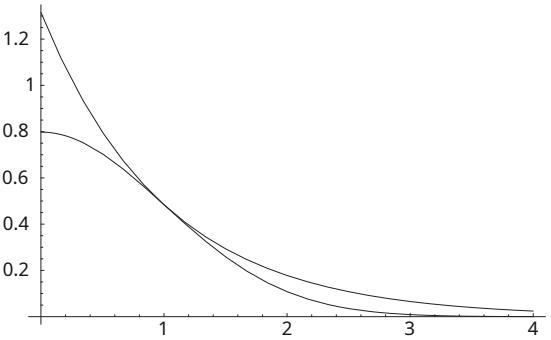

Figura 2.5: Acotando la densidad normal positiva.

La condición de aceptación, $tu \le f(x)/(C\mathrm{mi}^{-x})$, Se puede escribir como

$$tu \le \exp[-(X-1)^2/2], \tag{2.18}$$

que es equivalente a

$$-\ln tu \ge \frac{(X-1)^2}{2}, \tag{2.19}$$

dónde X es desde Exp(1). Ya que $-\ln tu$ también es de Exp(1), la última desigualdad se puede escribir como

$$V_1 \ge \frac{(V_2-1)^2}{2}, \tag{2.20}$$

dónde $V_1 = -\ln tu$ y $V_2 = X$ son independientes y ambos Exp(1) distribuido.

2.4.1.3 Distribución gamma Si $X \sim \mathrm{Gama}(\alpha, \lambda)$ entonces su pdf es de la forma

$$f(x) = \frac{x^{\alpha-1}\lambda^{\alpha}\mathrm{mi}^{-\lambda x}}{\Gamma(\alpha)}, \qquad x \ge 0. \tag{2.21}$$

Los parametros $\alpha > 0$ y $\lambda > 0$ se llaman los *forma* y *escala* parámetros, respectivamente. Ya que λ simplemente cambia la escala, es suficiente considerar solo la generación de variables aleatorias de $\mathrm{Gama}(\alpha, 1)$. En particular, si $X \sim \mathrm{Gama}(\alpha, 1)$, entonces $X/\lambda \sim \mathrm{Gama}(\alpha, \lambda)$ (ver Ejercicio 2.16). Debido a que el cdf para las distribuciones gamma generalmente no existe en forma explícita, el método de transformada inversa no siempre se puede aplicar para generar variables aleatorias a partir de esta distribución. Por lo tanto, se requieren métodos alternativos. Discutimos uno de esos métodos para el caso $\alpha > 1$. Deja $f(x) = x^{\alpha-1}\mathrm{mi}^{-x}/\Gamma(\alpha)$ y $\psi(x) = \alpha(1 + cx)^3, x > -1/c$ y cero en caso contrario, donde c y α son constantes positivas. Tenga en cuenta que $\psi(x)$ es una función estrictamente creciente. Dejar Y

tener densidad $k(y) = F(\psi(y))\psi'(y)C_1$, dónde C_1 es una constante de normalización. Después $X = \psi(Y)$ tiene densidad F. Es decir, por la regla de transformación (1.16), obtenemos

$$F_X(X) = \frac{k(\psi^{-1}(X))\ \psi'}{(\psi^{-1}(X))} = F(\psi(\psi^{-1}\quad(X)))\ \frac{\psi'(\psi^{-1}(X))\ \psi'}{(\psi^{-1}(X))} = F(X).$$

Dibujamos Y a través del método de aceptación-rechazo, usando la distribución normal estándar como nuestra propuesta de distribución. Nosotros elegimos C y d tal que $k(y) - C\phi(y)$, con $C > 1$ cerca de 1, donde ϕ es el pdf de la NORTE(0,1) distribución. Para encontrar tal C y d, primero escribimos $k(y) = C_2 \mathrm{mi}^{h(y)}$, donde un poco de álgebra mostrará que

$$h(y) = (1 - 3d)\ln(1 + cy) - d(1 + cy)^3 + d.$$

(Tenga en cuenta que $h(0) = 0$.) A continuación, una expansión en serie de Taylor de $h(y)$ alrededor de 0 rendimientos

$$h(y) = C(-1 - 3d + 3d)y - c_2\left(-1 + \frac{1}{2}6d + 3d\right)y_2 + O(y_3).$$

Esto sugiere tomar C y d tal que los coeficientes de y y y_2 en la expansión anterior son 0 y $-1/2$, respectivamente, como en el exponente de la densidad normal estándar. entonces tomamos $d = a - 1/3$ y $C = \frac{1}{\sqrt{1 + 3d}}$. No es difícil comprobar que entonces

$$h(y) \le -y_2 \text{ para todos } y > -\frac{1}{C},$$

y por lo tanto $e^{h(y)} \le \mathrm{mi}^{-1/2y_2}$. Lo que significa que $k(y)$ está dominado por $C_2\sqrt{2\pi}\phi(y)$ para todos y. Por lo tanto, el método de aceptación-rechazo para extraer de $Y \sim k$ es el siguiente: Dibujar $Z \sim$ NORTE(0,1) y $tu \sim$ tu(0,1) de forma independiente. Si

$$tu < \frac{C_2 \mathrm{mi}^{h(Z)}}{C_2\sqrt{2\pi}\,\phi(Z)},$$

o equivalentemente, si

$$\text{en } tu < h(Z) + Z_2/2,$$

luego regresa $Y = Z$; de lo contrario, repita (establecemos $h(Z) = -\infty$ si $Z = -1/C$). los eficiencia de este método, $\int_{-1/C}^{\infty} \mathrm{mi}^{h(y)}\,dtu / \int_{-1}^{\infty} \mathrm{mi}^{-1/2y_2}\,dy$, es mayor que 0.95 para todos valores de $a > 1$. Finalmente, completamos la generación de X tomando $X = \psi(Y)$. Para el caso en que $a < 1$, podemos usar el hecho de que si $X \sim$ Gama$(1 + a, 1)$ y $tu \sim$ tu(0,1) son independientes, entonces $XU^{1/a} \sim$ Gama$(a, 1)$; vea el Problema 2.17. Resumiendo, tenemos el siguiente algoritmo [15]:

Algoritmo 2.4.3: Muestreo de la Gama(α, λ) Distribución

función juego(α, λ)
  si $\alpha > 1$ **después**
    Establecer $d \leftarrow \alpha - 1/3$ y $C \leftarrow 1/\sqrt{9d}$
    Seguir $\leftarrow$ **verdadero**
    tiempo Seguir **hacer**
      Generar $Z \sim$ NORTE(0,1).
      si $Z > -1/C$ **después**
        $V \leftarrow (1 + cz)^3$
        Generar $tu \sim$ tu(0,1).
        si $tu < 1 - 0.0331Z^2 + dV + d\ln(V)$ **después** Seguir $\leftarrow$ **falso**
    $X \leftarrow dV/\lambda$
  más
    $X \leftarrow$ juego($\alpha+1, \lambda$)
    Generar $tu \sim$ tu(0,1). $X \leftarrow$
    $XU^{1/\alpha}$
  devolver X

Una distribución gamma con un parámetro de forma de número entero, digamos $\alpha = metro$, también se llama un *Distribución Erlang*, denotado Erl(m, λ). En este caso, X pueden $\sum$ representarse como la suma de iid variables aleatorias exponenciales Y_i. Eso es, $X = \sum_{i=1}^{metro} Y_i$, donde el $\{Y_i\}$ son variables exponenciales iid, cada una con media $1/\lambda$; ver Ejemplo 1.9. Usando Algoritmo 2.4.1, podemos escribir $Y_i = -\frac{1}{\lambda}\ln tu_i$, de donde

$$X = -\frac{1}{\lambda}\sum_{i=1}^{metro} \ln tu_i. \tag{2.22}$$

La ecuación (2.22) sugiere el siguiente algoritmo de generación:

Algoritmo 2.4.4: Generación de una variable aleatoria de Erlang

aporte : Entero positivo $metro$ y $\lambda > 0$.
producción: Variable aleatoria $X \sim$ Erl(m, λ).
Generar variables aleatorias iid $tu_1, \ldots, tu_{metro} \sim$ tu(0,1).
$X \leftarrow -\frac{1}{\lambda}\sum_{i=1}^{metro} \ln tu_i$
devolver X

2.4.1.4 Distribución beta Si $X \sim$ Beta(α, β), entonces su pdf es de la forma

$$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \qquad 0 < x < 1. \tag{2.23}$$

Ambos parámetros α y β se supone que son mayores que 0. Tenga en cuenta que Beta(1,1) es simplemente el tu(0,1) distribución.

Para tomar muestras de la distribución beta, consideremos primero el caso en el que α o β es igual a 1. En ese caso, simplemente podemos usar el método de la transformada inversa. por ejemplo, para $\beta = 1$, el Beta(α,1) pdf es

$$f(x) = \alpha x^{\alpha-1}, \qquad 0 < x < 1,$$

y la cdf correspondiente se convierte en

$$F(X) = X^{\alpha}, \qquad 0 < X < 1.$$

Así una variable aleatoria X se puede generar a partir de esta distribución dibujando $tu \sim \text{tu}(0,1)$ y regresando $X = tu^{1/\alpha}$.

Un procedimiento general para generar un Beta(α, β) variable aleatoria se basa en el hecho de que si $Y_1 \sim \text{Gama}(\alpha,1)$, $Y_2 \sim \text{Gama}(\beta,1)$, y Y_1 y Y_2 son independientes entonces

$$X = \frac{Y_1}{Y_1 + Y_2}$$

esta distribuido Beta(α, β). Se anima al lector a probar esta afirmación (vea el Problema 2.18). El algoritmo correspondiente es el siguiente:

Algoritmo 2.4.5: Generación de una variable aleatoria beta

aporte : $\alpha, \beta > 0$
producción: Variable aleatoria $X \sim \text{Beta}(\alpha, \beta)$.
1 generar independientemente $Y_1 \sim \text{Gama}(\alpha,1)$ y $Y_2 \sim \text{Gama}(\beta,1)$.
2 $X \leftarrow Y_1/(Y_1 + Y_2)$
3 **devolver** X

para entero $\alpha = metro$ y $\beta = norte$, se puede utilizar otro método, basado en la teoría de las estadísticas de orden. Dejar $tu_1, \ldots, tu_{metro+n-1}$ ser variables aleatorias independientes de tu(0,1). Entonces el $metro$ estadístico de -ésimo orden, $tu_{(metro)}$, tiene un Beta($m, norte$) distribución. Esto da el siguiente algoritmo.

Algoritmo 2.4.6: Generación de una variable aleatoria beta con parámetros enteros $\alpha = metro$ y $\beta = norte$

aporte : enteros positivos $metro$ y $norte$.
producción: Variable aleatoria $X \sim \text{Beta}(m, norte)$.
1 Generar $metro + norte - 1$ iid variables aleatorias $tu_1, \ldots, tu_{metro+n-1}$ de tu(0,1).
2 $X \leftarrow tu_{(metro)}$ // $metro$ estadístico de orden -th
3 **devolver** X

Se puede demostrar que el número total de comparaciones necesarias para encontrar $tu_{(metro)}$ es $(metro/2)(metro + 2norte - 1)$, por lo que este procedimiento pierde eficacia para grandes $metro$ y $norte$.

2.4.2 Generación de variables aleatorias discretas

2.4.2.1 Distribución de Bernoulli Si $X \sim \text{Ber}(pags)$, su pdf es de la forma

$$F(X) = pags^{x}(1 - pags)^{1-x}, \qquad X = 0,1, \tag{2.24}$$

dónde $pags$ es la probabilidad de éxito. Aplicando el método de la transformada inversa, podemos obtener fácilmente el siguiente algoritmo de generación:

Algoritmo 2.4.7: Generación de una variable aleatoria de Bernoulli

aporte : $pags \in (0,1)$

producción: Variable aleatoria $X \sim \text{Ber}(pags)$.

1 Generar $tu \sim \text{tu}(0,1)$.

2 **si** $tu < pags$ **después**

3 $X \leftarrow 1$

4 **más**

5 $X \leftarrow 0$

6 **devolver** X

En la Figura 2.6, se dan tres resultados típicos (realizaciones) para 100 variables aleatorias independientes de Bernoulli, cada una con un parámetro de éxito $pags=0.5$.

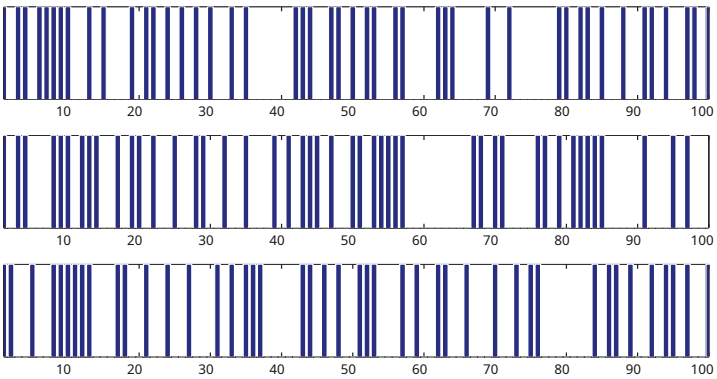

Figura 2.6: Resultados de tres experimentos con 100 ensayos de Bernoulli independientes, cada uno con $pags=0.5$. Las barras oscuras indican dónde aparece un éxito.

2.4.2.2 Distribución Binomial Si $X \sim$ Compartimiento($norte$, pag) entonces su pdf es de la forma

$$f(x) = \binom{norte}{x} pags^x (1-pags)^{n-x},\ x=0,1,\ldots,norte \tag{2.25}$$

Recuerde que una variable aleatoria binomial X puede verse como el número total de éxitos en $norte$ experimentos independientes de Bernoulli, cada uno con probabilidad de éxito $pags$; ver Ejemplo 1.1. Denota el resultado de la i-th juicio por $X_i=1$ (éxito) o $X_i=0$ (fracaso), podemos escribir $X = X_1+\cdots+X_{norte}$, con el $\{X_i\}$ siendo iidBer($pags$) variables aleatorias. Por lo tanto, el algoritmo de generación más simple se puede escribir de la siguiente manera:

Algoritmo 2.4.8: Generación de una variable aleatoria binomial

aporte : Entero positivo $norte$ y $pags \in (0,1)$. **producción:**

Variable aleatoria $X \sim$ Compartimiento($norte$, pag).

1 genero∑omió iid variables aleatorias $X_1, \ldots, X_{norte}$ deBer($pags$).

2 $X \leftarrow \sum_{i=1}^{norte} X_i$

3 **devolver** X

Dado que el tiempo de ejecución del Algoritmo 2.4.8 es proporcional a *norte*, podemos estar motivados para utilizar métodos alternativos para grandes *norte*. Por ejemplo, podríamos considerar la distribución normal como una aproximación a la binomial. En particular, por el teorema del límite central, como *norte* aumenta, la distribución de *X* está cerca de la de *Y* ~NORTE(*np, np*(1 −*pags*)); ver (1.26). De hecho, la cdf deNORTE(*np* -1⁄2, *notario*(1 −*pags*)) se aproxima a la cdf de *X* aun mejor. Esto se llama el *corrección de continuidad*.

Así, para obtener una variable aleatoria binomial, podríamos generar *Y* deNORTE(*np* - 1⁄2, *notario*(1 −*pags*)) y truncar al entero no negativo más próximo. De manera equivalente, podríamos generar *Z* ~NORTE(0,1) y establecer

$$\max\left\{0, \left\lfloor np + Z\sqrt{np(1-p)} + \tfrac{1}{2}\right\rfloor\right\} \tag{2.26}$$

como una muestra aproximada de laCompartimiento(*norte, pag*) distribución. Aquí $\lfloor a \rfloor$ denota la parte entera de *a*. Se debe considerar el uso de la aproximación normal para *np* >10 con *pags* ∈(0,1/2], y para *norte*(1 −*pags*)>10 con *pag* <1⁄2.

2.4.2.3 Distribución geométrica Si *X* ~GRAMO(*pags*), entonces su pdf es de la forma

$$f(x) = p(1-p)^{x-1}, \qquad x = 1, 2, \ldots \tag{2.27}$$

la variable aleatoria *X* puede interpretarse como el número de intentos necesarios hasta que se produce el primer éxito en una serie de intentos de Bernoulli independientes con parámetro de éxito *pags*. Tenga en cuenta quePAGS(*X* > *metro*) = (1 −*pags*)*metro*.

Ahora presentamos un algoritmo basado en la relación entre las distribuciones exponencial y geométrica. Dejar *Y* ~Exp(*λ*), con *λ* tal que 1 −*pags* = mi−*λ*. Después *X* = *Y* +1 tiene un GRAMO(*pags*) distribución. Esto es porque

$$\text{PAGS}(X > x) = \text{PAGS}(Y > x - 1) = \text{PAGS}(Y - X) = e^{-\lambda x} = (1-p)^x.$$

Por lo tanto, para generar una variable aleatoria a partir deGRAMO(*pags*), primero generamos una variable aleatoria a partir de la distribución exponencial con *λ* = −en(1 −*p*), trunca el valor obtenido al entero más cercano y suma 1.

Algoritmo 2.4.9: Generación de una variable aleatoria geométrica

aporte : *pags* ∈(0,1)

producción: Variable aleatoria *X* ~GRAMO(*pags*).

1 Generar *Y* ~Exp(−en(1 −*pags*)).

2 $X \leftarrow 1 + Y$

3 **devolver** *X*

2.4.2.4 Distribución de Poisson Si *X* ~Poi(*λ*), su pdf es de la forma

$$f(x) = \frac{e^{-\lambda}\lambda^{x}}{x!}, \qquad x = 0, 1, \ldots, \tag{2.28}$$

dónde *λ* es el *Velocidad* parámetro. Existe una relación íntima entre Poisson y variables aleatorias exponenciales, destacadas por las propiedades del proceso de Poisson; consulte la Sección 1.12. En particular, una variable aleatoria de Poisson *X* puede interpretarse como el número máximo de variables exponenciales iid (con parámetro *λ*)

cuya suma no exceda de 1. Es decir,

$$X = \text{máximo}\left\{ norte : \sum_{j=1}^{norte} Y_j \le 1 \right\}, \tag{2.29}$$

donde el $\{Y_j\}$ son independientes y Exp(λ) distribuido. Ya que $Y_j = -1/\lambda \ln tu_j$, con $tu_j \sim$ tu(0,1), podemos reescribir (2.29) como

$$\begin{aligned} X &= \text{máximo}\left\{ norte : \sum_{j=1}^{norte} -\ln tu_j \le \lambda \right\} \\ &= \text{máximo}\left\{ norte : \ln\left(\prod_{j=1}^{norte} tu_j \right) \ge -\lambda \right\} \\ &= \text{máximo}\left\{ norte : \prod_{j=1}^{norte} tu_j \ge \text{mi}^{-\lambda} \right\}. \end{aligned} \tag{2.30}$$

Esto conduce al siguiente algoritmo:

Algoritmo 2.4.10: Generación de una variable aleatoria de Poisson

aporte : $\lambda > 0$

producción: Variable aleatoria $X \sim$ Poi(λ).

1 Establecer $norte \leftarrow 0$ y $a \leftarrow 1$.
2 **tiempo** $a \ge \text{mi}^{-\lambda}$ **hacer**
3 Generar $tu \sim$ tu(0,1). $a \leftarrow$
4 *una tu*
5 $norte \leftarrow norte + 1$
6 $X \leftarrow norte - 1$
7 **devolver** X

Se ve fácilmente que para grandes λ, este algoritmo se vuelve lento ($\prod_{j=1}^{norte} tu_j < \text{mi}^{-\lambda}$ es pequeño para largo λ, y más números aleatorios, tu_j, son necesarios para satisfacer). Enfoques alternativos se pueden encontrar en [2] y [7].

2.5 GENERACIÓN DE VECTORES ALEATORIOS

Digamos que necesitamos generar un vector aleatorio $\mathbf{X} = (X_1, \ldots, X_{norte})$ de un dado $norte$-distribucion dimensional con pdf $f(\mathbf{x})$ y CDF $F(\mathbf{x})$. Cuando los componentes $X_1, \ldots, X_{norte}$ son *independiente*, la situación es fácil: simplemente aplicamos el método de la transformada inversa u otro método de generación de nuestra elección a cada componente individualmente.

EJEMPLO 2.9

Queremos generar vectores aleatorios uniformes $\mathbf{X} = (X_1, \ldots, X_{norte})$ desde el $norte$-rectángulo dimensional $D = \{(X_1, \ldots, X_{norte}) : a_i \le X_i \le b_i, i = 1, \ldots, n\}$. Está

claro que los componentes de $\mathbf{X}$ son independientes y uniformemente distribuidas: $X_i \sim tu[a_i, b_i]$, $i = 1, \ldots, norte$. Aplicando el método de la transformada inversa a X_i, por lo tanto podemos escribir $X_i = a_i + (b_i - a_i)tu_i$, $i = 1, \ldots, n$, dónde $tu_1, \ldots, tu_{norte}$ son iid detu(0,1).

Para *dependiente* variables aleatorias $X_1, \ldots, X_{norte}$, podemos representar el pdf conjunto $f(\mathbf{X})$, usando la regla del producto (1.4), como

$$f(X_1, \ldots, X_{norte}) = f_1(X_1) f_2(X_2 \mid X_1) \cdots f_{norte}(X_{norte} \mid X_1, \ldots, X_{n-1}), \tag{2.31}$$

dónde $f_1(X_1)$ es la fdp marginal de X_1 y $f_k(X_k \mid X_1, \ldots, X_{k-1})$ es el pdf condicional de X_k dado $X_1 = x_1, X_2 = x_2, \ldots, X_{k-1} = x_{k-1}$. Por lo tanto, una forma de generar $\mathbf{X}$ es generar primero X_1, entonces, dado $X_1 = x_1$, para generar X_2 de $f_2(X_2 \mid x_1)$, y así sucesivamente, hasta generar X_{norte} de $f_{norte}(X_{norte} \mid x_1, \ldots, x_{n-1})$.

La aplicabilidad de este enfoque depende, por supuesto, del conocimiento de las distribuciones condicionales. En ciertos modelos, como los modelos de Markov, este conocimiento se puede obtener fácilmente.

2.5.1 Método de aceptación-rechazo de vectores

El Algoritmo de aceptación-rechazo 2.3.6 es directamente aplicable al caso multidimensional. Solo debemos tener en cuenta que la variable aleatoria x (ver Línea 3 del Algoritmo 2.3.6) se convierte en un *norte*-vector aleatorio dimensional $\mathbf{X}$. En consecuencia, necesitamos una forma conveniente de generar $\mathbf{X}$ de la propuesta multidimensional pdf *gramo*($\mathbf{X}$), por ejemplo, utilizando el método de transformada inversa vectorial. El siguiente ejemplo demuestra la versión vectorial del método de aceptación-rechazo.

EJEMPLO 2.10

Queremos generar un vector aleatorio $\mathbf{Z}$ que se distribuye uniformemente sobre una superficie irregular *norte*-región dimensional *GRAMO* (ver Figura 2.7). El algoritmo es sencillo:

1. *Generar un vector aleatorio, $\mathbf{X}$, distribuida uniformemente en W, dónde W es una región regular (hipercubo multidimensional, hiperrectángulo, hiperesfera, hiperelipsoide, etc.).*

2. *Si $\mathbf{X} \in GRAMO$, aceptar $\mathbf{Z} = \mathbf{X}$ como el vector aleatorio distribuido uniformemente sobre GRAMO; de lo contrario, regrese al Paso 1.*

Como caso especial, dejemos *GRAMO* ser el *norte*-unidad dimensional bola, es decir, $\sum_i x_{2i} \le 1$ - *GRAMO* = $\{\mathbf{X} : 1\}$, y deja W ser el *norte*-hipercubo dimensional $\{-1 \le x_i \le 1\}_{i=1}^{norte}$. para generar un vector aleatorio que se distribuye uniformemente en el interior del *norte*-bola unitaria -dimensional, generamos un vector aleatorio $\mathbf{X}$ que se distribuye uniformemente en W y luego aceptarlo o rechazarlo, según quede dentro o fuera del *norte*-bola dimensional. El algoritmo correspondiente es el siguiente:

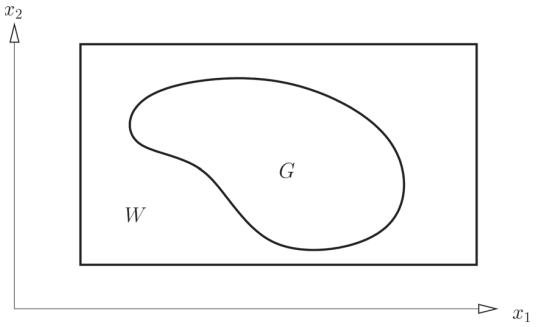

Figura 2.7: El método de aceptación-rechazo del vector.

Algoritmo 2.5.1: Generación de un Vector Aleatorio Uniformemente Distribuido Dentro del *norte*-Bola de unidad dimensional

producción: vector aleatorio **X** en la bola unidad.

1 $R \leftarrow \infty$
2 **tiempo** $R > 1$ **hacer**
3 Generar $tu_1, \ldots, tu_{norte}$ como iid variables aleatorias detu(0,
4 1). $X_1 \leftarrow \sum -2tu_1, \ldots, X_{norte} \leftarrow 1 - 2tu_{norte}$
5 $R \leftarrow \sum_{i=1}^{norte} X_i^2$
6 $\mathbf{X} \leftarrow (X_1, \ldots, X_{norte})$
7 **volver X**

Observación 2.5.1 Para generar un vector aleatorio que se distribuye uniformemente sobre el *superficie* o $\sum$ admirador *norte*-bola unitaria dimensional — en otras palabras, uniformemente sobre la unidad esfera $\{\mathbf{X}: \sum_i X_i^2 = 1\}$ —nosotros necesitamos solo para escalar el vector **X** tal que tiene unidad longitud. Es decir, volvemos $\mathbf{Z} = \mathbf{X} / R$ en vez de **X**.

La eficiencia del método de aceptación-rechazo del vector es igual a la relación

$$\frac{1}{C} = \frac{\text{volumen de la hiperbola}}{\text{volumen del hipercubo}} = \frac{1}{norte 2^{n-1} \Gamma(norte/2)} \frac{\pi^{norte/2}}{},$$

donde los volúmenes de la bola y el cubo son $\frac{\pi^{norte/2}}{(norte/2)\Gamma(norte/2)}$ y 2^{norte}, respectivamente. Nota que incluso para *norte* (*norte* = 2*metro*) tenemos

$$\frac{1}{C} = \frac{\pi^{metro}}{metro!\, 2^{2metro}} = \frac{1}{metro!} \frac{(\pi)^{metro}}{2} \quad 2^{-m} \to 0 \text{ como } metro \to \infty.$$

En otras palabras, el método de aceptación-rechazo se vuelve ineficiente en *norte*, y es asintóticamente inútil.

2.5.2 Generación de variables a partir de una distribución multinormal

La clave para generar un vector aleatorio normal multivariado (o simplemente multinormal) $\mathbf{Z} \sim$ NORTE($\boldsymbol{m}, \Sigma$) es escribirlo como $\mathbf{Z} = \boldsymbol{m} + B\mathbf{X}$, dónde B una matriz tal que *cama y desayuno* $= \Sigma$,

y$\mathbf{X}$es un vector de iidNORTE(0,1) variables aleatorias; consulte la Sección 1.10. Tenga en cuenta que $\boldsymbol{m} = (m_1, \ldots, m_{norte})$ es el vector medio y Σ es el ($norte \times norte$) matriz de covarianza de$\mathbf{Z}$. Para cualquier matriz de covarianza Σ, tal matriz$B$siempre se puede encontrar de manera eficiente utilizando el método de la raíz cuadrada de Cholesky; véase la Sección A.1 del Apéndice.

El siguiente algoritmo describe la generación de unNORTE($\boldsymbol{m}$,Σ)vector aleatorio distribuido$\mathbf{Z}$:

Algoritmo 2.5.2: Generación de Vectores Multinormales

aporte : Vector medio$\boldsymbol{m}$y matriz de covarianza Σ.

producción:vector aleatorio$\mathbf{Z}$~NORTE($\boldsymbol{m}$,Σ).

1Generar$X_1, \ldots, X_{norte}$como variables iid deNORTE(0,1).

2Deduzca la descomposición inferior de Cholesky Σ =*cama y desayuno*.

3Establecer$\mathbf{Z} \leftarrow \boldsymbol{m} + B\mathbf{X}$.

4**volver Z**

2.5.3 Generación de vectores aleatorios uniformes sobre un símplex

Considera el$norte$-dimensional símplex,

$$Y = \{\mathbf{y} : y_i \geq 0, \quad i = 1, \ldots, n, \quad \sum_{i=1}^{norte} y_i \leq 1\}. \tag{2.32}$$

Yes un simplex en los puntos$\mathbf{0}, \mathbf{mi}_1, \ldots, \mathbf{mi}_{norte}$, dónde$\mathbf{0}$es el vector cero y$\mathbf{mi}_i$es el i-ésimo vector unitario enR$_{norte}$,$i = 1, \ldots, norte$. Dejar$X$ser un segundo$norte$-simplex dimensional:

$$X = \{\mathbf{X} : X_i \geq 0,\ i = 1, \ldots, n,\ x_1 - X_2 - \cdots - X_{norte} \leq 1\}.$$

Xes un simplex en los puntos$\mathbf{0}, \mathbf{mi}_{norte}, \mathbf{mi}_{norte} + \mathbf{mi}_{n-1}, \ldots, \mathbf{1}$, dónde$\mathbf{1}$es la suma de todos los vectores unitarios (un vector de 1s). La figura 2.8 ilustra el caso bidimensional.

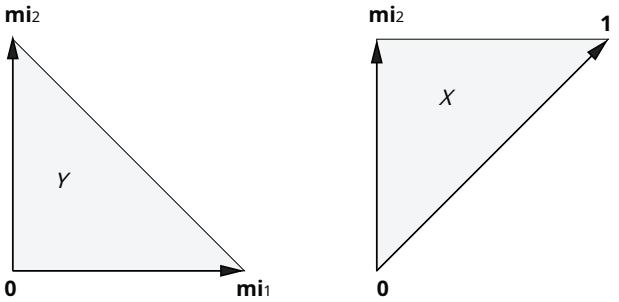

Figura 2.8: Símplex$Y$yX.

símplex$Y$se puede obtener de simplex$X$por la transformación lineal$\mathbf{y} = A\mathbf{X}$ con

$$A = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -1 & 1 \end{pmatrix}.$$

Ahora, dibujando un vector $\mathbf{X} = (X_1, \ldots, X_{norte})$ de acuerdo con la distribución uniforme en X es fácil: simplemente toma X_i ser el iEstadístico de -ésimo orden de variables aleatorias iid $tu_1, \ldots, tu_{norte}$detu(0,1). Dado que una transformación lineal conserva la uniformidad, la aplicación de la matriz Aa $\mathbf{X}$produce un vector $\mathbf{Y}$que se distribuye uniformemente en Y.

Algoritmo 2.5.3: Generación de un vector sobre una unidad simplex Y

producción: vector aleatorio $\mathbf{Y}$distribuido uniformemente sobre el símplex Y.

1 Generar $norte$variables aleatorias independientes $tu_1, \ldots, tu_{norte}$detu(0,1).

2 Clasificar $tu_1, \ldots, tu_{norte}$en las estadísticas de pedidos $tu_{(1)}, \ldots, tu_{(norte)}$.

3 $Y_1 \leftarrow tu_{(1)}$

4 **por** $i = 2$ **a** $norte$ **hacer** $Y_i = tu_{(i)} - tu_{(yo-1)}$

5 $\mathbf{Y} \leftarrow (Y_1, \ldots, Y_{norte})$

6 **volver Y**

si definimos $Y_{norte+1} = 1 - \sum_{i=1}^{norte} Y_i = 1 - U_{(norte)}$, entonces la resultante ($norte$+1)-dimensional vectores $(Y_1, \ldots, Y_{norte+1})$ se distribuirá uniformemente en el conjunto

$$F = \left\{ \mathbf{y} : y_i \geq 0, \quad i = 1, \ldots, norte+1, \quad \sum_{i=1}^{norte+1} y_i = 1 \right\},$$

es decir, sobre la cara dominante del símplex definido por los puntos $\mathbf{0}, \mathbf{mi}_1, \ldots, \mathbf{mi}_{norte+1}$.

Finalmente, para generar vectores aleatorios distribuidos uniformemente sobre un $norte$-símplex dimensional definido por vértices arbitrarios, digamos $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{norte}$, simplemente generamos $\mathbf{Y}$uniformemente encendido Yy aplicar la transformación lineal

$$\mathbf{Z} = C\mathbf{Y} + \mathbf{z}_0,$$

dónde Ces la matriz cuyas columnas son $\mathbf{z}_1 - \mathbf{z}_0, \ldots, \mathbf{z}_{norte} - \mathbf{z}_0$.

2.5.4 Generación de vectores aleatorios distribuidos uniformemente sobre una hiperbola e hiperesfera unitarias

El algoritmo 2.5.1 y la observación 2.5.1 explican cómo, utilizando el método multidimensional de aceptación-rechazo, se pueden generar vectores aleatorios que se distribuyen uniformemente en un $norte$-unidad dimensional hiperbola (o simplemente $norte$-pelota). Simplemente dividiendo cada vector por su longitud, se obtienen vectores aleatorios que se distribuyen uniformemente sobre el *superficie* del $norte$-pelota, es decir, la $norte$-esfera. La principal ventaja del método de aceptación-rechazo es su simplicidad. Su principal desventaja es que el número de intentos necesarios para generar puntos dentro del $norte$-la bola aumenta explosivamente con $norte$. Por esta razón, puede recomendarse solo para dimensiones bajas ($norte$-5). Un algoritmo alternativo se basa en el siguiente resultado:

Teorema 2.5.1 *Dejar* $X_1, \ldots, X_{norte}$ *ser iid variables aleatorias de* NORTE(0,1)*, y deja* $||\mathbf{X}|| = (\sum_{i=1}^{norte} X_{21})^{2}$*. Entonces el vector*

$$\mathbf{Y} = \left(\frac{X_1}{||\mathbf{X}||}, \ldots, \frac{X_{norte}}{||\mathbf{X}||} \right) \tag{2.33}$$

se distribuye uniformemente sobre la $norte$*-esfera* $\{\mathbf{y} : ||\mathbf{y}|| = 1\}$.

Prueba: Tenga en cuenta que **Y** es simplemente la proyección de **X** = ($X_1, \ldots, X_{norte}$) sobre la *norte*-esfera. El hecho de que **Y** se distribuye uniformemente se sigue inmediatamente del hecho de que la fdp de **X** es esféricamente simétrica: $f_X(\mathbf{X}) = C \exp(-\|\mathbf{x}\|^2/2)$. -

Para obtener variables aleatorias uniformes dentro del *norte*-bola, simplemente multiplicamos el vector **Y** por $tu^{1/norte}$, dónde $tu \sim$ tu(0,1). Para ver esto, tenga en cuenta que para un vector aleatorio **Z** = ($Z_1, \ldots, Z_{norte}$) que se distribuye uniformemente en el *norte*-bola, el radio $R = \|\mathbf{Z}\|$ satisface $P(R \le r) = r^{norte}$. Por lo tanto, por el método de la transformada inversa, podemos escribir $R = tu^{1/norte}$. Esto motiva la siguiente alternativa:

Algoritmo 2.5.4: Generación de vectores aleatorios uniformes dentro del *norte*-Pelota

producción: vector aleatorio **Z** distribuida uniformemente dentro de la *norte*-pelota.

1 Generar un vector aleatorio **X** = ($X_1, \ldots, X_{norte}$) con iid NORTE(0,1) componentes.
2 Generar $R = tu^{1/norte}$, con $tu \sim$ tu(0,1).
3 **Z** $\leftarrow R\mathbf{X}/\|\mathbf{X}\|$
4 **volver Z**

2.5.5 Generación de vectores aleatorios uniformemente distribuidos dentro de un hiperelipsoide

La ecuación para un hiperelipsoide, centrado en el origen, se puede escribir como

$$\mathbf{X}\Sigma\mathbf{X} = r^2, \tag{2.34}$$

donde Σ es un definido positivo y simétrico (*norte* × *norte*) matriz (**X** se interpreta como un vector columna). El caso especial donde Σ = *yo* (matriz identidad) corresponde a una hiperesfera de radio *r*. Dado que Σ es definida positiva y simétrica, existe una única matriz triangular inferior *B* tal que Σ = *cama y desayuno*; ver (1.25). Así podemos ver el conjunto $X = \{\mathbf{X} : \mathbf{X}\Sigma\mathbf{X} - r^2\}$ como una transformación lineal **y** = *B***X** del *norte*-bola dimensional $Y = \{\mathbf{y} : \mathbf{y}\mathbf{y} - r^2\}$. Dado que las transformaciones lineales conservan la uniformidad, si el vector **Y** se distribuye uniformemente en el interior de un *norte*-esfera dimensional de radio *r*, entonces el vector **X** = (*B* distribuida sobre el interior de un hiperelipsoide (ver (2.34)). algoritmo de generación se da a continuación.) = **Y** es uniformemente el correspondiente

Algoritmo 2.5.5: Generación de vectores aleatorios uniformes en un hiperelipsoide

aporte : Σ, $r > 0$

producción: vector aleatorio **X** distribuida uniformemente dentro del hiperelipsoide.

1 Generar **Y** = ($Y_1, \ldots, Y_{norte}$) distribuidos uniformemente dentro de la *norte*-bola de radio *r*.
2 Calcular la matriz (de Cholesky inferior) *B*, satisfaciendo Σ = *cama y desayuno*.
3 **X** $\leftarrow (B)^{-1}$**Y**
4 **volver X**

2.6 PROCESOS GENERADORES DE VENENOS

Esta sección trata la generación de procesos de Poisson. Recuerde de la Sección 1.12 que hay dos caracterizaciones diferentes (pero equivalentes) de un proceso de Poisson

{NORTE$_t$, t-0}. En el primero (ver Definición 1.12.1), el proceso se interpreta como una medida de conteo, donde *norte*$_t$ cuenta el número de llegadas en [0, t]. La segunda caracterización es que los tiempos entre llegadas *{A$_i$}* de *{norte$_t$, t-0}* formar un *proceso de renovación*, es decir, una secuencia de variables aleatorias iid. En este caso, la entrada los tiempos tienen un Exp(λ) distribución, y podemos escribir $A_i = -1 \lambda \text{en} tu_i$, donde el *{tu$_i$}* son iidtu(0,1) distribuido. Usando la segunda caracterización, podemos generar los tiempos de llegada $T = A_1 + \cdots + A_i$ durante el intervalo [0, T] como sigue:

Algoritmo 2.6.1: Generación de un proceso de Poisson homogéneo

aporte : Tiempo final T, Velocidad $\lambda > 0$.

producción: Número de llegadas *norte* y tiempos de llegada $T_1, \ldots, T_{norte}$.

1 Establecer $T_0 \leftarrow 0$ y *norte* $\leftarrow 0$.
2 **tiempo** $T_{norte} < T$ **hacer**
3 Generar $tu \sim$ tu(0,1).
4 $T_{norte+1} \leftarrow T_{norte} - 1\lambda \text{en} tu$
5 *norte* $\leftarrow$ *norte* + 1
6 **devolver** *norte*, $t_1, \ldots, T_{norte}$

La primera caracterización de un proceso de Poisson, es decir, como una medida de conteo aleatorio, proporciona una forma alternativa de generar tales procesos, que funciona también en el caso multidimensional. En particular (vea el final de la Sección 1.12), el siguiente procedimiento puede usarse para generar un proceso de Poisson homogéneo con tasa λ en cualquier conjunto A con "volumen" $|A|$:

Algoritmo 2.6.2: Generando un *norte*-Proceso de veneno dimensional

aporte : Velocidad $\lambda > 0$.

producción: Número de puntos *norte* y puestos $\mathbf{X}_1, \ldots, \mathbf{X}_{norte}$.

1 Generar una variable aleatoria de Poisson *norte* $\sim$ Poi($\lambda |A|$).
2 Dibujar *norte* puntos $\mathbf{X}_1, \ldots, \mathbf{X}_{norte}$ independiente y uniformemente en A.
3 **volver** $\mathbf{X}_1, \ldots, \mathbf{X}_{norte}$

A *proceso de Poisson no homogéneo* es un proceso de conteo *norte* = *{NORTE$_t$, t-0}* para el cual el número de puntos en intervalos que no se superponen son independientes, similar al proceso de Poisson ordinario, pero la velocidad a la que llegan los puntos es *dependiente del tiempo*. Si $\lambda(t)$ denota la tasa en el tiempo t, el número de puntos en cualquier intervalo (*antes de Cristo*) tiene una distribución de Poisson con media $\int_a^b \lambda(t)\, dt$.

La Figura 2.9 ilustra una forma de construir tales procesos. Primero generamos un proceso de Poisson homogéneo bidimensional en la tira *{(t, x), t-0,0 -X-λ}*, con tasa constante $\lambda = \text{máx.} \lambda(t)$, y luego simplemente proyecte todos los puntos debajo del gráfico de $\lambda(t)$ sobre la t-eje.

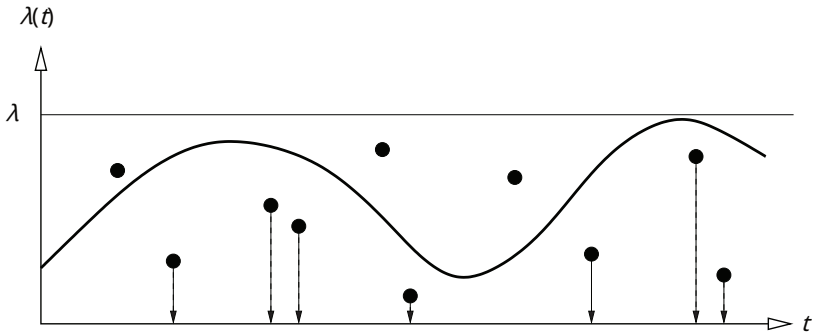

Figura 2.9: Construcción de un proceso de Poisson no homogéneo.

Tenga en cuenta que los puntos del proceso de Poisson bidimensional pueden verse como si tuvieran una dimensión de tiempo y espacio. Las épocas de llegada forman un proceso de Poisson unidimensional con tasa λ, y las posiciones son uniformes en el intervalo $[0, l]$. Esto sugiere el siguiente procedimiento alternativo para generar procesos de Poisson no homogéneos: cada época de llegada del Poisson homogéneo unidimensional el proceso es rechazado (adelgazado) con probabilidad $1 - \lambda(T_{norte}) / \lambda$, dónde T_{norte} es la llegada tiempo de la *norte*-la ventilación. Las épocas supervivientes definen el proceso de Poisson no homogéneo deseado.

Algoritmo 2.6.3: Generación de un proceso de Poisson no homogéneo

aporte : Tiempo final T y tasa de función $\lambda(t)$, con máx. $\lambda(t) = \lambda$. **producción:** Número de llegadas *norte* y tiempos de llegada $T_1, T_2, \ldots, T_{norte}$.

```
t←0 y norte←0
tiempo t < T hacer
    Generar tu~tu(0,1).
    t←t -1  λ en tu
    Generar V~tu(0,1). si V-
    λ(t)/λ después
        T_norte+1 ←t
        norte←norte+1
devolver norte y T1, . . . , T_norte
```

2.7 GENERACIÓN DE PROCESOS DE CADENAS DE MARKOV Y SALTOS DE MARKOV

Ahora discutimos cómo simular una cadena de Markov $X_0, X_1, X_2, \ldots, X_{norte}$. Para generar una cadena de Markov con distribución inicial $\boldsymbol{\pi}_{(0)}$ y matriz de transición *PAGS*, podemos usar el procedimiento descrito en la Sección 2.5 para variables aleatorias dependientes. Es decir, primero generar X_0 de $\boldsymbol{\pi}_{(0)}$. Entonces, dado $X_0 = X_0$, generar X_1 de la distribución condicional de X_1 dado $X_0 = X_0$; en otras palabras, generar X_1 desde el X_0-ésima fila de *PAGS*. Suponer $X_1 = X_1$. Entonces, genera X_2 desde el X_1-primera fila de *PAGS*, y así. El algoritmo para una cadena de Markov de estado discreto general con una matriz de transición de un paso *PAGS* y un vector de distribución inicial $\boldsymbol{\pi}_{(0)}$ es como sigue:

Algoritmo 2.7.1: Generación de una cadena de Markov

aporte : Tamaño de la muestra $norte$, distribución inicial $\boldsymbol{\pi}_{(0)}$, matriz de transición $PAGS$.
producción: cadena de Markov $X_0, \ldots, X_{norte}$.

1 Dibujar X_0 de la distribución inicial $\boldsymbol{\pi}_{(0)}$.
2 **por** t=1 **a** $norte$ **hacer**
3 Dibujar X_t de la distribución correspondiente a la X_{t-1}-ésima fila de $PAGS$.
4 **devolver** $X_0, \ldots, X_{norte}$

EJEMPLO 2.11 Paseo aleatorio sobre los enteros

Considere la caminata aleatoria sobre los enteros en el ejemplo 1.10. Dejar X_0=0 (es decir, empezamos en 0). Supongamos que la cadena está en algún momento discreto t=0,1,2. . . en estado i. Luego, en la Línea 3 del Algoritmo 2.7.1, simplemente necesitamos dibujar a partir de una distribución de dos puntos con masa $pags$ y q a i+1 y yo -1, respectivamente. En otras palabras, dibujamos yo_t ~Ber($pags$) y establecer $X_{t+1}=X_t+2yo_t-1$. La Figura 2.10 da una ruta de muestra típica para el caso donde $pags=q=1/2$.

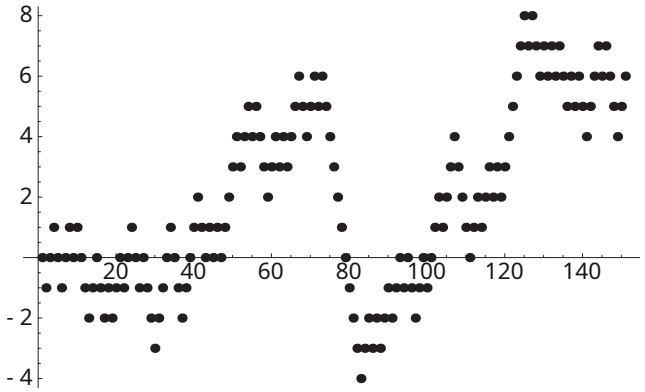

Figura 2.10: Paseo aleatorio sobre los enteros, con $pags=q=1/2$.

2.7.1 Paseo aleatorio en un gráfico

Como generalización del ejemplo 2.11, podemos asociar una caminata aleatoria con cualquier gráfica $GRAMO$, cuyo espacio de estado es el conjunto de vértices del gráfico y cuyas probabilidades de transición de i a j son iguales a $1/d_i$, dónde d_i es el grado de i (el número de aristas de i). Una propiedad importante de estos paseos aleatorios es que son reversibles en el tiempo. Esto se puede verificar fácilmente a partir del criterio de Kolmogorov (1.39). En otras palabras, no hay un "bucle" sistemático. En consecuencia, si la gráfica es conexa y si la distribución estacionaria $\{\pi_i\}$ existe, que es el caso cuando el gráfico es finito, entonces las ecuaciones de balance local se mantienen:

$$\pi_i pags_{yo} = \pi_j pags_{ji}. \tag{2.35}$$

Cuando $pags_{yo} = pags_{yo}$ para todos i y j, se dice que la caminata aleatoria es *simétrico*. De (2.35) se sigue inmediatamente que en este caso la distribución de equilibrio es uniforme en el espacio de estados mi.

EJEMPLO 2.12 Paseo aleatorio simple en un *norte*-Cubo

Queremos simular un paseo aleatorio sobre los vértices de la *norte*-hipercubo dimensional (o simplemente *norte*-cubo); véase la figura 2.11 para el caso tridimensional.

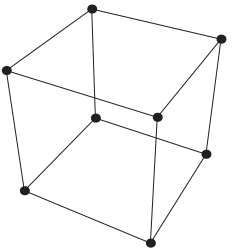

Figura 2.11: En cada paso, se elige al azar uno de los tres vecinos del vértice actualmente visitado.

Tenga en cuenta que los vértices de la *norte*-cubo son de la forma $\mathbf{X} = (X_1, \ldots, X_{norte})$, con $X0$ o 1. El conjunto de los 2_{norte} de estos vértices se denota $\{0,1\}_{norte}$. Generamos un paseo aleatorio $\{X_t,\ t=0,1,2,\ldots\}$ en $\{0,1\}_{norte}$ como sigue: Sea el estado inicial X_0 ser arbitrario, digamos $X_0 = (0, \ldots, 0)$. Dado $X_t = (X_{t1}, \ldots, X_{Tennesse})$, elige aleatoriamente una coordenada j según la distribución uniforme discreta en el conjunto $\{1, \ldots n\}$. Si j es el resultado, luego reemplace X_{jn} con $1 - X_{jn}$. Al hacerlo, obtenemos en la etapa $t+1$,

$$X_{t+1} = (X_{t1}, \ldots, 1 - X_{tj}, X_{t(j+1)}, \ldots, X_{Tennesse}),$$

y así.

2.7.2 Generación de procesos de salto de Markov

La generación de procesos de salto de Markov es bastante similar a la generación de cadenas de Markov de arriba. Suponer $X = \{X_t,\ t-0\}$ es un proceso de salto de Markov con tasas de transición $\{q_{yo}\}$. De la Sección 1.13.5, recuerde que el proceso de salto de Markov salta de un estado a otro según una cadena de Markov $Y = \{Y_{norte}\}$ (la cadena de salto), y el tiempo pasado en cada estado i se distribuye exponencialmente con un parámetro que puede depender de i. La matriz de transición de un paso, digamos k, de Y y los parametros $\{q_{ij}\}$ de t∑ Los tiempos de espera exponenciales se pueden encontrar directamente desde el $\{q_{yo}\}$. A saber, $q_i = {}_j q_{yo}$ (la suma de las tasas de transición de i), y $k(yo, j) = q_{yo}/q_i$ por $i = j$ (por lo tanto, las probabilidades son simplemente proporcionales a las tasas). Tenga en cuenta que $k(yo, yo) = 0$. Definiendo los tiempos de espera como $A_1, A_2, \ldots$ y los tiempos de salto como $T_1, T_2, \ldots$, podemos escribir el algoritmo ahora de la siguiente manera:

Algoritmo 2.7.2: Generación de un proceso de salto de Markov

aporte : Tiempo final T, distribución inicial $\boldsymbol{\pi}^{(0)}$, tasas de transición $\{q_{yo}\}$. **producción:** Número de saltos $norte$, tiempos de salto $T_1, \ldots, T_{norte}$ y estados de salto $Y_0, \ldots, Y_{norte}$.

1 Dibujar Y_0 de la distribución inicial $\boldsymbol{\pi}^{(0)}$.
2 Inicializar $X_0 \leftarrow Y_0$, $T_0 \leftarrow 0$, y $norte \leftarrow 0$.
3 **tiempo** $T_{norte} < T$ **hacer**
4 Dibujar $A_{norte+1}$ de $\text{Exp}(q_{Y_{norte}})$
5 $T_{norte+1} \leftarrow T_{norte} + A_{norte+1}$
6 Establecer $X_t \leftarrow Y_{norte}$ por $T_{norte} \le t < T_{norte+1}$
7 Dibujar $Y_{norte+1}$ de la distribución correspondiente a la Y_{norte}-ésima fila de k.
8 $norte \leftarrow norte + 1$
9 **devolver** $norte, t_1, \ldots, T_{norte}, Y_0, Y_1, \ldots, Y_{norte}$

2.8 GENERACIÓN DE PROCESOS GAUSSIANOS

Recuerde de la Sección 1.14 que, en un proceso gaussiano $\{X_t, t \in T\}$, cada subvector $(X_{t_1}, \ldots, X_{t_{norte}})$ tiene una distribución multinormal, NORTE($\boldsymbol{m}, \Sigma$), para algún vector de expectativa $\boldsymbol{m}$, y la matriz de covarianza Σ, dependiendo de $t_1, \ldots, t_{norte}$. En particular, si el proceso gaussiano tiene una función de expectativa $m_t, t \in T$ y función de covarianza $\Sigma_{s\,t}, s\ t \in T$, después $\boldsymbol{m} = (m_{t_1}, \ldots, m_{t_{norte}})$ y $\Sigma_{yo} = \Sigma_{t_i, t_j}, yo, j = 1, \ldots, norte$. En consecuencia, el algoritmo 2.5.2 se puede utilizar para generar realizaciones de un proceso gaussiano en tiempos específicos (índices) $t_1, \ldots, t_{norte}$.

Aunque siempre es posible encontrar una descomposición de Cholesky $\Sigma = cama\ y\ desayuno$ (ver Sección A.1 del Apéndice), este procedimiento requiere en general $O(norte^3)$ operaciones de punto flotante (para un $norte \times norte$ matriz). Como resultado, la generación de vectores gaussianos de alta dimensión requiere mucho tiempo para grandes $norte$, a menos que el proceso tenga una estructura adicional.

EJEMPLO 2.13 Generación de procesos Wiener

En el ejemplo 1.14, el proceso de Wiener se definió como un proceso gaussiano de media cero $\{X_t, t \ge 0\}$ con función de covarianza $\Sigma_{s\,t} = s$ para $0 \le s \le t$.

Evidentemente, no es posible generar la ruta completa de la muestra, ya que habría que generar una cantidad infinita de variables. Sin embargo, podríamos usar el Algoritmo 2.5.2 para generar resultados del vector aleatorio normal $\mathbf{X} = (X_{t_1}, \ldots, X_{t_{norte}})$ por tiempos $t_1 < \cdots < t_{norte}$. La matriz de covarianza Σ de $\mathbf{X}$ es en este caso

$$\Sigma = \begin{pmatrix} t_1 & t_1 & t_1 & \ldots & t_1 \\ t_1 & t_2 & t_2 & \ldots & t_2 \\ t_1 & t_2 & t_3 & \ldots & t_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_1 & t_2 & t_3 & \ldots & t_{norte} \end{pmatrix}.$$

Fácilmente podemos verificar (multiplicar B con B es) que la matriz de Cholesky inferior B dado por

$$B=\begin{pmatrix} \sqrt{t_1} & 0 & 0 & \dots & 0 \\ \sqrt{t_1} & \sqrt{t_2-t_1} & \sqrt{t_3-t_2} & \dots & 0 \\ \sqrt{t_1} & \sqrt{t_2-t_1} & \sqrt{t_3-t_2} & \dots & \sqrt{t_{norte}-t_{n-1}} \end{pmatrix}.$$

Esto nos lleva al siguiente algoritmo de simulación:

Algoritmo 2.8.1: Generación de un proceso Wiener

aporte : Veces $0 = t_0 < t_1 < t_2 < \cdots < t_{norte}$.

producción: Wiener procesa variables aleatorias a veces $t_0, \dots, t_{norte}$.

1 $X_0 \leftarrow 0$
2 **por** $k = 1$ **a** *norte* **hacer**
3 Dibujar $Z \sim$ NORTE(0,1)
4 $X_{t_k} \leftarrow X_{t_{k-1}} + \sqrt{t_k - t_{k-1}}\, Z$
5 **devolver** $X_{t_0}, \dots, X_{t_{norte}}$

La figura 1.8 en la página 28 da una realización típica de un proceso de Wiener en el intervalo [0,1], evaluado en puntos de cuadrícula $t_{norte} = norte\,10^{-5}$, $norte = 0,1, \dots, 10^5$.

Al utilizar las propiedades del proceso de Wiener (vea el Ejemplo 1.14), podemos justificar el Algoritmo 2.8.1 de una manera más directa. Específicamente, X_{t_1} tiene un distribución NORTE$(0, t_1)$ y, por lo tanto, puede generarse como $\sqrt{t_1}\,Z_1$, dónde Z_1 tiene una distribución normal estándar. Además, para cada $k = 2,3, \dots, norte$, la variable aleatoria X_{t_k} es igual a $X_{t_{k-1}} + tu_t$ donde el incremento tu_t es independiente de $X_{t_{k-1}}$ y tiene un NORTE$(t_k - t_{k-1})$ distribución, y se puede generar como $\sqrt{t_k - t_{k-1}}\,Z_k$.

2.9 GENERACIÓN DE PROCESOS DE DIFUSIÓN

El proceso de Wiener, indicado en esta sección por $\{W_t,\ t \ge 0\}$, juega un papel importante en la probabilidad y forma la base de los llamados *procesos de difusión*, denotado aquí por $\{X_t,\ t \ge 0\}$. Estos son procesos de Markov con un parámetro de tiempo continuo y con rutas de muestra continuas, como el proceso de Wiener.

Un proceso de difusión a menudo se especifica como la solución de un *ecuación diferencial estocástica* (SDE), que es una expresión de la forma

$$\mathrm{d}X_t = a(X_t, t)\,\mathrm{d}t + b(X_t, t)\,\mathrm{d}W_t, \tag{2.36}$$

dónde $\{W_t,\ t \ge 0\}$ es un proceso de Wiener y $a(x, t)$ y $b(x, t)$ son funciones deterministas. El coeficiente (función) a se llama el *deriva*, y b^2 se llama el *difusión* coeficiente. Cuando a y b son constantes, digamos $a(x, t) = m$ y $b(x, t) = \sigma$, el proceso de difusión resultante es de la forma

$$X_t = mt + \sigma W_t$$

y se llama un *movimiento browniano* con deriva m y coeficiente de difusión σ^2.

Una técnica simple para simular aproximadamente los procesos de difusión es *método de euler*; véase, por ejemplo, [4]. La idea es reemplazar la SDE con la ecuación de diferencia estocástica

$$Y_{k+1} = Y_k + a(Y_k, kh)h + b(Y_k, kh)h\sqrt{}Z_{k+1}, k = 0,1,2, \ldots, \tag{2.37}$$

dónde $Z_1, Z_2, \ldots$ son independientes NORTE(0,1) variables aleatorias distribuidas. Para un tamaño de paso pequeño h, el proceso $\{Y_k, k = 0,1,2, \ldots\}$ aproxima el proceso $\{X_t, t \geq 0\}$ en el sentido de que $Y_k \approx X_{kh}, k = 0,1,2, \ldots$.

Algoritmo 2.9.1: Método de Euler para SDE

aporte : Numero de pie h, tamaño de la muestra *norte*, funciones a y b.
producción: Aproximación del proceso de difusión SDE en tiempos 0, $h, 2h, \ldots, No$.
1 Generar Y_0 de la distribución de X_0.
2 **por** $k = 1$ **a** *norte* **hacer**
3 Dibujar $Z \sim$ NORTE(0,1)
4 $Y_{k+1} \leftarrow Y_k + a(Y_k, kh)h + b(Y_k, kh)h\sqrt{}Z$
5 **devolver** $Y_0, \ldots, Y_{norte}$

EJEMPLO 2.14 Movimiento Browniano Geométrico

El movimiento browniano geométrico se usa a menudo en ingeniería financiera para modelar el precio de un activo riesgoso; véase, por ejemplo, [3]. El SDE correspondiente es

$$dX_t = mX_t dt + \sigma X_t dW_t,$$

con valor inicial X_0. La aproximación de Euler es

$$Y_{k+1} = Y_k\left(1 + mh + \sigma\sqrt{}h\, Z_{k+1}\right), k = 0,1,2, \ldots,$$

dónde $Z_1, Z_2, \ldots$ son variables aleatorias normales estándar independientes.

La figura 2.12 muestra una trayectoria típica del movimiento browniano geométrico que comienza en $X_0 = 1$, con parámetros $m = 1$ y $\sigma = 0.2$, generado a través de la aproximación de Euler con tamaño de paso $h = 10^{-4}$. La línea discontinua es la gráfica de $t \to \text{Exp}[t(\mu - \sigma^2/2)]$ a lo largo del cual fluctúa el proceso.

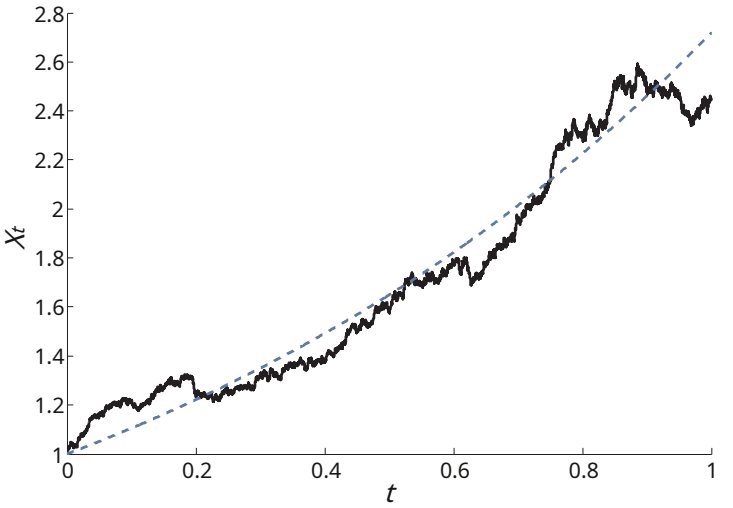

Figura 2.12: Movimiento browniano geométrico.

Se pueden encontrar métodos de aproximación más elaborados para SDE, por ejemplo, en [4].

2.10 GENERACIÓN DE PERMUTACIONES ALEATORIAS

Muchos algoritmos de Monte Carlo implican la generación de permutaciones aleatorias, es decir, el orden aleatorio de los números 1,2, . . . , *norte*, para algunos fijos *norte*. Para ver ejemplos de problemas interesantes asociados con la generación de permutaciones aleatorias, vea el problema del vendedor ambulante en el Capítulo 6, el problema permanente en el Capítulo 9 y el Ejemplo 2.15 a continuación.

Supongamos que queremos generar cada uno de los *norte*! ordenamientos posibles con igual probabilidad. Presentamos dos algoritmos para lograr esto. El primero se basa en la ordenación de una secuencia de *norte* números aleatorios uniformes. En el segundo, elegimos los componentes de la permutación consecutivamente. El segundo algoritmo es más rápido que el primero.

Algoritmo 2.10.1: Primer algoritmo para generar permutaciones aleatorias

aporte : Entero *norte*-1.

producción: Permutación aleatoria ($X_1, \ldots, X_{norte}$) de (1, . . . , *norte*).

1 Generar $tu_1, tu_2, \ldots, tu_{norte}$ ~tu(0,1) de forma independiente.

2 Ordena estos en orden creciente.

3 Dejar ($X_1, \ldots, X_{norte}$) sean los índices de los sucesivos valores ordenados.

4 **devolver** ($X_1, \ldots, X_{norte}$)

Por ejemplo, deja *norte*=4 y supongamos que los números generados (tu_1, tu_2, tu_3, tu_4) son (0.7,0.3,0.5,0.4). Ya que (tu_2, tu_4, tu_3, tu_1) = (0.3,0.4,0.5,0.7) es la secuencia ordenada, la permutación resultante es (2,4,3,1). El inconveniente de este algoritmo.

es que requiere ordenar una secuencia de $norte$ números aleatorios, lo que requiere $norte$ en $norte$ comparaciones

Como mencionamos, el segundo algoritmo se basa en la idea de generar los componentes de la permutación aleatoria uno por uno. El primer componente se elige al azar (con igual probabilidad) de $1, \ldots, norte$. El siguiente componente se elige aleatoriamente de los números restantes, y así sucesivamente. Por ejemplo, deja $norte$=4. Extraemos el componente 1 de la distribución uniforme discreta en $\{1,2,3,4\}$. Digamos que obtenemos 2. Nuestra permutación es entonces de la forma $(2, \cdot, \cdot, \cdot)$. A continuación generamos a partir de la distribución uniforme de tres puntos en $\{1,3,4\}$. Ahora, digamos que se elige 1. Así, nuestro resultado intermedio para la permutación es $(2,1, \cdot, \cdot)$. Por último, para el tercer componente, podemos elegir 3 o 4 con igual probabilidad. Supongamos que dibujamos 4. La permutación resultante es (2,1,4,3). Generando una variable aleatoria X de una distribución uniforme discreta en $\{X_1, \ldots, X_k\}$ se hace eficientemente generando primero $yo = k\ tu + 1$, con $tu \sim$ tu(0,1) y regresando $X = X_{yo}$. Así tenemos el siguiente algoritmo:

Algoritmo 2.10.2: Segundo algoritmo para generar permutaciones aleatorias

aporte : Entero $norte$-1.

producción: Permutación aleatoria $(X_1, \ldots, X_{norte})$ de $(1, \ldots, norte)$.

1 Establecer $PAGS = \{1, \ldots n\}$.

2 **por** i=1 **a** $norte$ **hacer**

3 Generar X_i de la distribución uniforme discreta en $PAGS$.

4 Remover X_i de $PAGS$.

5 **devolver** $(X_1, \ldots, X_{norte})$

Observación 2.10.1 Para aumentar aún más la eficiencia del segundo algoritmo de permutación aleatoria, podemos implementarlo de la siguiente manera: Sea **pags** = $(pags_i, \ldots, pags_{norte})$ sea un vector que almacene los resultados intermedios del algoritmo en el i-ésimo paso. Inicialmente, deja **pags** = $(1, \ldots, norte)$. Dibujar X_1 seleccionando uniformemente un índice $yo \in \{1, \ldots, n\}$, y volver $X_1 = pags_{yo}$. Después, $intercambio\ X_1$ y $pags_{norte} = norte$. En el segundo paso, dibuja X_2 seleccionando uniformemente yo de $\{1, \ldots, n-1\}$, devolver $X_2 = pags_{yo}$ y cambiarlo con $pags_{n-1}$, y así. De esta forma, el algoritmo requiere la generación de sólo $norte$ números aleatorios uniformes (para extraer de $\{1,2, \ldots, k\}$, $k = norte, norte$ -1, $\ldots$,2) y $norte$ operaciones de permuta.

EJEMPLO 2.15 Generación de un recorrido aleatorio en un gráfico

Considere un gráfico ponderado $GRAMO$ con $norte$ nodos, etiquetados $1,2, \ldots, norte$. Los nodos representan ciudades y los bordes representan los caminos entre las ciudades. El problema es generar aleatoriamente un *recorrido* que visita todas las ciudades exactamente una vez excepto la ciudad de inicio, que también es la ciudad de destino. Sin pérdida de generalidad, supongamos que el gráfico es completo, es decir, todas las ciudades están conectadas. Podemos representar cada recorrido a través de una permutación de los números $1, \ldots, norte$. por ejemplo, para $norte$=4, la permutación (1,3,2 ,4) representa el recorrido $1 \to 3 \to 2 \to 4 \to 1$.

De manera más general, representamos un recorrido a través de una permutación. $\mathbf{X} = (X_1, \ldots, X_{norte})$ con $X_1 = 1$, es decir, asumimos sin pérdida de generalidad que comenzamos el recorrido en la ciudad número 1. Para generar un recorrido aleatorio uniformemente en X, podemos

simplemente aplique el Algoritmo 2.10.2. Tenga en cuenta que el número de todos los recorridos posibles de elementos en el conjunto de todos los recorridos posibles$X$es

$$|X| = (norte - 1)! \tag{2.38}$$

PROBLEMAS

2.1 Aplique el método de la transformada inversa para generar una variable aleatoria a partir de la distribución uniforme discreta con pdf

$$F(X) = \begin{cases} \frac{1}{norte+1}, & X=0,1,\ldots,n, \text{ de lo} \\ 0 & \text{contrario.} \end{cases}$$

2.2 Explique cómo generar a partir de laBeta(1, β) distribución utilizando el método de la transformada inversa.

2.3 Explique cómo generar a partir de laWeib(α, λ) distribución utilizando el método de la transformada inversa.

2.4 Explique cómo generar a partir de laPareto(α, λ) distribución utilizando el método de la transformada inversa.

2.5 Muchas familias de distribuciones son de*escala de ubicación*escribe. Es decir, la cdf tiene la forma

$$F(X) = F_0\left(\frac{x-m}{\sigma}\right),$$

dónde$m$se llama el*ubicación*parámetro yσla*escala*parámetro, yF_0es un cdf fijo que no depende de$m$yσ. losNORTE(μ, σ^2) familia de distribuciones, dondeF_0es el cdf normal estándar, es un ejemplo básico. Escribe$F(X;\mu, \sigma)$por$F(X)$. Dejar$X\sim F_0$(es decir,$X\sim F(X; 0, 1)$). Pruebalo$Y=m+\sigma X\sim F(X;\mu, \sigma)$. Por lo tanto, para tomar muestras de cualquier cdf en una familia de escala de ubicación, es suficiente saber cómo tomar muestras deF_0.

2.6 Aplicar el método de la transformada inversa para generar variables aleatorias a partir de un *Distribución de Laplace*(es decir, una distribución exponencial de dos colas desplazada) con pdf

$$F(X) = \frac{\lambda}{2}e^{-\lambda|x-\theta|}, \qquad -\infty < X < \infty (\lambda > 0).$$

2.7 Aplique el método de la transformada inversa para generar una variable aleatoria a partir de la *distribución de valores extremos*, que tiene CDF

$$F(X) = 1 - \exp(-\exp(\tfrac{x-\mu}{\sigma})), \quad -\infty < X < \infty (\sigma > 0).$$

2.8 Considere la variable aleatoria triangular con pdf

$$F(X) = \begin{cases} 0 & \text{si} X < 2a \text{ o } X > 2b, \\ \frac{x-2a}{(segundo - un)^2} & \text{si } 2a - x < un + b, \\ \frac{(2segundo - x)}{(segundo - un)^2} & \text{si} a + b - X < 2b. \end{cases}$$

a) Derive el cdf correspondiente F.

b) Demuestre que aplicando el método de la transformada inversa se obtiene

$$X = \begin{cases} 2a + (segundo - un)\sqrt{2tu} & \text{si } 0 < tu < \tfrac{1}{2}, \\ 2b + (un - segundo)\sqrt{2(1-tu)} & \text{si } \tfrac{1}{2} < tu < 1. \end{cases}$$

2.9 Presentar un algoritmo de transformada inversa para generar una variable aleatoria a partir del pdf constante por partes

$$f(x) = \begin{cases} C_i, & x_{i-1} < x < x_i,\ i = 1, 2, \ldots, n, \text{ de lo} \\ 0 & \text{contrario,} \end{cases}$$

dónde $C_i > 0$ y $x_0 < x_1 < \cdots < x_{n-1} < x_{norte}$.

2.10 Dejar

$$f(x) = \begin{cases} C_i x, & x_{i-1} < x < x_i, \text{ de lo} \quad i = 1, \ldots, n, \\ 0 & \text{contrario,} \end{cases}$$

dónde $C_i > 0$ y $x_0 < x_1 < \cdots < x_{n-1} < x_{norte}$.

a) Dejar $F_i = \sum_{j=1}^{i} \int_{x_{j-1}}^{x_j} C_j t u \, dtu, i = 1, \ldots, norte$. Demostrar que la cdf F satisface

$$F(x) = F_{i-1} + \frac{C_i}{2}\left(x^2 - x_{i-1}^2\right), \qquad x_{i-1} < x < x_i,\ i = 1, \ldots, norte$$

b) Describir un algoritmo de transformada inversa para la generación de variables aleatorias a partir de $f(x)$.

2.11 Se dice que una variable aleatoria tiene un *cauchy* distribución si su pdf está dada por

$$f(x) = \frac{1}{\pi}\frac{1}{1 + x^2}, \quad x \in \mathbb{R}. \tag{2.39}$$

Explique cómo se pueden generar variables aleatorias de Cauchy utilizando el método de la transformada inversa.

2.12 Si X y Y son variables aleatorias normales estándar independientes, entonces $Z = X/Y$ tiene una distribución de Cauchy. Muestre esto. [Sugerencia: Primero demuestre que si tu y $V > 0$ son rand continuos ∫om variables con pdf conjunto $f_{ultravioleta}$, entonces el pdf de $W = U/V$ es dado por $f_W(w) = \int_0^\infty f_{ultravioleta}(wv, v) v \, dv$.]

2.13 Verificar la validez de la composición Algoritmo 2.3.4.

2.14 Utilizando el método de composición, formule e implemente un algoritmo para generar variables aleatorias a partir de la siguiente mezcla normal (gaussiana) pdf:

$$f(x) = \sum_{i=1}^{3} \frac{1}{b_i}\, pags_i\, \phi\left(\frac{x - un_i}{b_i}\right)$$

dónde ϕ es el pdf de la distribución normal estándar y $(pags_1, pags_2, pags_3) = (1/2, 1/3, 1/6)$, $(a_1, a_2, a_3) = (-1, 0, 1)$, y $(b_1, b_2, b_3) = (1/4, 1, 1/2)$.

2.15 Comprueba eso $C = 2e\sqrt{/\pi}$ en la figura 2.5.

2.16probar que si$X$~Gama(α,1), entoncesX/λ~Gama(α, λ).

2.17 Dejar$X$~Gama(1 +α,1) ytu~tu(0,1) ser independiente. Siα <1, entonces $XU^{1/a}$~ Gama(α,1). Demuestra esto.

2.18 SiY_1 ~Gama(α,1), Y_2 ~Gama(β,1), yY_1yY_2son independientes entonces

$$X = \frac{Y_1}{Y_1 + Y_2}$$

esBeta(α, β) distribuido. Demuestra esto.

2.19 Diseñe un algoritmo de aceptación-rechazo para generar una variable aleatoria a partir del pdf$F$dado en (2.17) usando unExp(λ) distribución de propuestas. Cualλda la mayor probabilidad de aceptación?

2.20 El pdf de la distribución exponencial truncada con parámetroλ= 1 está dado por

$$F(X) = \frac{\text{mi}^{-x}}{1 - \text{mi}^{-a}}, \quad 0 < X < una .$$

a)Diseñe un algoritmo para generar variables aleatorias a partir de esta distribución utilizando el método de la transformada inversa.

b)Construya un algoritmo de generación que use el método de aceptación-rechazo con unExp(λ) distribución de propuestas.

C)Encuentre la eficiencia del método de aceptación-rechazo para los casos$a$=1, y $a$acercándose a cero y al infinito.

2.21 Sea la variable aleatoria$X$tener pdf

$$F(X) = \begin{cases} \frac{1}{4}, & 0 < X < 1, \\ x - 3/4, & 1 < X < 2. \end{cases}$$

Generar una variable aleatoria a partir de$F(X)$, usando

a)el método de la transformada inversa,

b)el método de aceptación-rechazo, utilizando la densidad de propuesta

$$gramo(x) = \frac{1}{2}, 0 < X < 2.$$

2.22 Sea la variable aleatoria$X$tener pdf

$$F(X) = \begin{cases} \frac{1}{2}X, & 0 < X < 1, \\ \frac{1}{2}, & 1 < X < 5/2. \end{cases}$$

Generar una variable aleatoria a partir de$F(X)$, usando

a)el método de la transformada inversa

b)el método de aceptación-rechazo, utilizando la densidad de propuesta

$$gramo(X) = \frac{8}{25}X, 0 < X < \frac{5}{2} .$$

2.23 Dejar$X$tener una distribucion geometrica truncada, con pdf

$$F(X) = c.p.(1 - pags)^{x-1}, \; X = 1, \ldots, norte ,$$

dónde$C$es una constante de normalización. Generar una variable aleatoria a partir de$F(X)$, usando

a)el método de la transformada inversa,

b)el método de aceptación-rechazo, conGRAMO($pags$) como la distribución de la propuesta. Encuentre la eficiencia del método de aceptación-rechazo para$norte$=2 y $norte$=∞.

2.24 Generar una variable aleatoria$Y$=min$_{i=1,\ldots,metro}$máximo$_{j=1,\ldots,r}\{X_{yo}\}$, suponiendo que las variables$X_{yo,}$ i=1, . . . , $m,$ j=1, . . . r, son iid con cdf común$F(X)$, utilizando el método de la transformada inversa. [Sugerencia: use los resultados para la distribución de estadísticas de pedidos en el Ejemplo 2.5.]

2.25generar 100Ber(0.2) variables aleatorias tres veces y producir gráficos de barras similares a los de la figura 2.6. repetir paraBer(0.5).

2.26 Genere un proceso de Poisson homogéneo con tasa 100 en el intervalo [0,1]. Use esto para generar un proceso de Poisson no homogéneo en el mismo intervalo, con función de tasa

$$\lambda(t) = 100 \text{ pecado}_2(10t), \quad t\text{-}0.$$

2.27 Genere y grafique una realización de los puntos de un proceso de Poisson bidimensional con tasaλ= 2 en el cuadrado [0,5]x[0,5]. ¿Cuántos puntos caen en el cuadrado [1,3]x[1,3]? ¿Cuántos esperas que caigan en este cuadrado?

2.28 Escriba un programa que genere y muestre 100 vectores aleatorios que estén uniformemente distribuidos dentro de la elipse

$$5x^2+21xy+25y^2=9.$$

2.29 Implemente ambos algoritmos de permutación aleatoria en la Sección 2.10. Compara su desempeño.

2.30 Considere una caminata aleatoria en el gráfico no dirigido de la figura 2.13. Por ejemplo, si la caminata aleatoria en algún momento está en el estado 5, saltará a 3, 4 o 6 en la siguiente transición, cada uno con una probabilidad de 1/3.

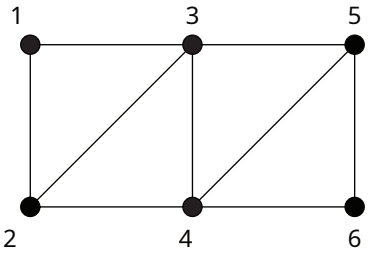

Figura 2.13: Un gráfico.

a)Encuentre la matriz de transición de un paso para esta cadena de Markov.

b)Demuestre que la distribución estacionaria está dada por$\boldsymbol{\pi}$= (1 9,16,29,29,16,19).

C)Simule la caminata aleatoria en una computadora y verifique que, a la larga, la proporción de visitas a los distintos nodos está de acuerdo con la distribución estacionaria.

2.31 Genere varias rutas de muestra para la caminata aleatoria en los enteros para $p = 1/2$ y $p = 2/3$.

2.32 Considera el $M/M/1$ sistema de colas del Ejemplo 1.13. Dejar X_t sea el número de clientes en el sistema en ese momento t. Escribir un programa de computadora para simular el proceso estocástico $X = \{X_t\}$ viendo X como un proceso de salto de Markov, y aplicando el Algoritmo 2.7.2. Presentar rutas de muestra del proceso para los casos. $\lambda = 1$, $m = 2$ y $\lambda = 10$, $m = 11$

Otras lecturas

Las referencias clásicas sobre generación de números aleatorios y generación de variables aleatorias son [5] y [2]. Otras referencias incluyen [8], [14] y [18] y el tutorial en [17]. Una buena referencia es [1]. La simulación de procesos espaciales se discute en [6].

REFERENCIAS

1. S. Asmussen y PW Glynn. *Simulación estocástica*. Springer-Verlag, Nueva York, 2007.

2. L. Devroye. *Generación de variantes aleatorias no uniformes*. Springer-Verlag, Nueva York, 1986.

3. P. Glasserman. *Métodos Monte Carlo en Ingeniería Financiera*. Springer-Verlag, Nueva York, 2004.

4. PE Kloeden y E. Platen. *Solución numérica de ecuaciones diferenciales estocásticas*. Springer-Verlag, Nueva York, 1992. Tercera edición corregida.

5. DE Knuth. *El arte de la programación informática*, volumen 2: *Algoritmos Semiméricos*. Addison-Wesley, Reading, MA, 2ª edición, 1981.

6. DP Kroese y ZI Botev. Simulación de procesos espaciales. En V. Schmidt, editor, *Conferencias sobre Geometría Estocástica, Estadística Espacial y Campos Aleatorios*, volumen II: Análisis, Modelado y Simulación de Estructuras Complejas. Springer, Berlín, 2014.

7. DP Kroese, T. Taimre y ZI Botev. *manual de métodos de monte carlo*. John Wiley & Sons, 2011.

8. AM Law y WD Kelton. *Modelado y análisis de simulación*. McGraw-Hill, Nueva York, 3ª edición, 2000.

9. P. L'Ecuyer. Buenos parámetros e implementaciones para múltiples generadores de números aleatorios recursivos combinados. *La investigación de operaciones*, 47(1):159-164, 1999.

10. P. L'Ecuyer y F. Panneton. F_2-Generadores de números aleatorios lineales. En C. Alexopoulos, D. Goldsman y JR Wilson, editores, *Avanzando en las fronteras de la simulación: un festival en honor a George Samuel Fishman*, páginas 175–200, Nueva York, 2009. Springer-Verlag.

11. P. L'Ecuyer y R. Simard. TestU01: Biblioteca AC para pruebas empíricas de generadores de números aleatorios. *Transacciones ACM en software matemático*, 33(4), 2007. Artículo 22.

12. DH Lehmer. Métodos matemáticos en unidades de computación a gran escala. *Anales del Laboratorio de Computación de la Universidad de Harvard*, 26:141–146, 1951.

13. PA Lewis, AS Goodman y JM Miller. Un generador de números pseudoaleatorios para el sistema/360. *Diario de sistemas de IBM*, 8(2):136–146, 1969.

14. NN Madrás. *Conferencias sobre los métodos de Monte Carlo.* Sociedad Matemática Estadounidense, 2002.

15. G. Marsaglia y W. Tsang. Un método simple para generar variables gamma. *Transacciones ACM en software matemático*, 26(3):363–372, 2000.

16. M. Matsumoto y T. Nishimura. Mersenne twister : un generador de números pseudoaleatorios uniformes equidistribuidos de 623 dimensiones. *Transacciones ACM en Modelado y Simulación por Computadora*, 8(1):3–30, 1998.

17. BD Ripley. Generación informática de variables aleatorias: un tutorial. *Revista Estadística Internacional*, 51:301–319, 1983.

18. SM Ross. *Simulación*. Academic Press, Nueva York, 3ª edición, 2002.

19. AJ Walker. Un método eficiente para generar variables aleatorias discretas con distribuciones generales. *Transacciones ACM en software matemático*, 3:253–256, 1977.

CAPÍTULO 3

SIMULACIÓN DE SISTEMAS DE EVENTOS DISCRETOS

3.1 INTRODUCCIÓN

La simulación por computadora ha servido durante mucho tiempo como una herramienta importante en una amplia variedad de disciplinas: ingeniería, investigación de operaciones y ciencia administrativa, estadística, matemáticas, física, economía, biología, medicina, ingeniería, química y ciencias sociales. A través de la simulación por computadora, se puede estudiar el comportamiento de los sistemas de la vida real que son demasiado difíciles de examinar analíticamente. Se pueden encontrar ejemplos en vuelos de aviones supersónicos, sistemas de comunicaciones telefónicas, pruebas en túneles de viento, gestión de batallas a gran escala (p. ej., para evaluar sistemas de armas defensivos u ofensivos) u operaciones de mantenimiento (p. ej., para determinar el tamaño óptimo de las cuadrillas de reparación), por mencionar algunos. Avances recientes en metodologías de simulación, disponibilidad de software, análisis de sensibilidad, y la optimización estocástica se han combinado para hacer de la simulación una de las herramientas más aceptadas y utilizadas en el análisis de sistemas y la investigación de operaciones. El crecimiento sostenido en tamaño y complejidad de los sistemas emergentes del mundo real (p. ej., redes de comunicación de alta velocidad y sistemas biológicos) sin duda garantizará que la popularidad de la simulación por computadora continúe creciendo.

El objetivo de este capítulo es proporcionar una breve introducción al arte y la ciencia de la simulación por computadora, en particular con respecto a los sistemas de eventos discretos. El capítulo está organizado de la siguiente manera: la Sección 3.2 describe conceptos básicos como sistemas, modelos, simulación y métodos de Monte Carlo. La Sección 3.3 trata de los ingredientes más fundamentales de la simulación de eventos discretos, a saber, el reloj de simulación y la lista de eventos. La Sección 3.4 explica las ideas detrás de la simulación de eventos discretos a través de una serie de ejemplos resueltos.

3.2 MODELOS DE SIMULACIÓN

por un *sistema* nos referimos a una colección de entidades relacionadas, a veces llamadas *componentes* o *elementos*, formando un todo complejo. Por ejemplo, un hospital puede considerarse un sistema, con médicos, enfermeras y pacientes como elementos. Los elementos poseen ciertas características o *atributos* que toman valores lógicos o numéricos. En nuestro ejemplo, un atributo puede ser el número de camas, el número de máquinas de rayos X, el nivel de habilidad, etc. Por lo general, las actividades de los componentes individuales interactúan con el tiempo. Estas actividades provocan cambios en el estado del sistema. Por ejemplo, el estado de la sala de espera de un hospital puede describirse por el número de pacientes que esperan a un médico. Cuando un paciente llega al hospital o sale de él, el sistema salta a un nuevo estado.

Nos ocuparemos únicamente de *sistemas de eventos discretos*, a saber, aquellos sistemas en los que las variables de estado cambian instantáneamente a través de saltos en puntos discretos en el tiempo, a diferencia de *sistemas continuos*, donde las variables de estado cambian continuamente con respecto al tiempo. Ejemplos de sistemas discretos y continuos son, respectivamente, un banco que atiende a clientes y un automóvil que circula por la autopista. En el primer caso, el número de clientes en espera es una variable de estado constante por tramos que cambia solo cuando llega un nuevo cliente al banco o cuando un cliente termina de realizar sus transacciones comerciales y sale del banco; en el último caso, la velocidad del automóvil es una variable de estado que puede cambiar continuamente con el tiempo.

El primer paso para estudiar un sistema es construir un modelo a partir del cual obtener predicciones sobre el comportamiento del sistema. por un *modelo* nos referimos a una abstracción de algún sistema real que puede usarse para obtener predicciones y formular estrategias de control. A menudo, estos modelos son de naturaleza matemática (fórmulas, relaciones) o gráfica. Por lo tanto, el sistema físico real se traduce, a través del modelo — en un sistema matemático. Para ser útil, un modelo debe necesariamente incorporar elementos de dos características contrapuestas: realismo y simplicidad. Por un lado, el modelo debe proporcionar una aproximación razonablemente cercana al sistema real e incorporar la mayoría de los aspectos importantes del sistema real. Por otro lado, el modelo no debe ser tan complejo como para impedir su comprensión y manipulación.

Hay varias formas de evaluar la validez de un modelo. Por lo general, comenzamos a probar un modelo volviendo a examinar la formulación del problema y descubriendo posibles fallas. Otra verificación de la validez de un modelo es asegurarse de que todas las expresiones matemáticas sean dimensionalmente consistentes. Una tercera prueba útil consiste en variar los parámetros de entrada y verificar que la salida del modelo se comporte de manera plausible. La cuarta prueba es la llamada *retrospectivo* prueba. Implica el uso de datos históricos para reconstruir el pasado y luego determinar qué tan bien se habría desempeñado la solución resultante si se hubiera utilizado. La comparación de la efectividad de este desempeño hipotético con lo que realmente sucede indica qué tan bien el modelo predice la realidad. Sin embargo, una desventaja de las pruebas retrospectivas es que utiliza los mismos datos que el modelo. A menos que el pasado sea una réplica representativa del futuro, es mejor no recurrir a esta prueba en absoluto.

Una vez que se ha construido un modelo para el sistema en cuestión, el siguiente paso es derivar una solución de este modelo. Para ello, ambos *analítico* y *numérico* Se pueden invocar métodos de solución. Una solución analítica generalmente se obtiene directamente de su representación matemática en forma de fórmulas. Una solución numérica es generalmente una aproximación a través de un procedimiento de aproximación adecuado.

Gran parte de este libro trata de soluciones numéricas y métodos de estimación obtenidos mediante simulación por computadora. Más precisamente, usamos *Simulación estocástica por computadora* - llamado a menudo *simulación del Monte Carlo*—que incluye cierta aleatoriedad en el modelo subyacente, en lugar de una simulación informática determinista. El término *Monte Carlo* fue utilizado por von Neumann y Ulam durante la Segunda Guerra Mundial como palabra clave para el trabajo secreto en Los Álamos sobre problemas relacionados con la bomba atómica. Ese trabajo involucró la simulación de la difusión aleatoria de neutrones en materiales nucleares.

Naylor et al. [7] definir *simulación* como sigue:

> *La simulación es una técnica numérica para realizar experimentos en una computadora digital, que involucra ciertos tipos de modelos lógicos y matemáticos que describen el comportamiento de los sistemas comerciales o económicos (o algún componente de los mismos) durante un período prolongado de tiempo real.*

La siguiente lista de situaciones típicas debería dar al lector una idea de dónde la simulación sería una herramienta apropiada.

- El sistema puede ser tan complejo que una formulación en términos de una ecuación matemática simple puede ser imposible. La mayoría de los sistemas económicos entran en esta categoría. Por ejemplo, a menudo es prácticamente imposible describir el funcionamiento de una empresa comercial, una industria o una economía en términos de unas pocas ecuaciones simples. Otra clase de problemas que conduce a dificultades similares es el de los sistemas de colas complejos a gran escala. La simulación ha sido una herramienta sumamente eficaz para tratar problemas de este tipo.

- Incluso si se puede formular un modelo matemático que capture el comportamiento de algún sistema de interés, puede que no sea posible obtener una solución al problema incorporado en el modelo mediante técnicas analíticas sencillas. Una vez más, los sistemas económicos y los complejos sistemas de colas ejemplifican este tipo de dificultad.

- La simulación se puede utilizar como un dispositivo pedagógico para enseñar tanto a los estudiantes como a los profesionales las habilidades básicas en análisis de sistemas, análisis estadístico y toma de decisiones. Entre las disciplinas en las que se ha utilizado con éxito la simulación con este fin se encuentran la administración de empresas, la economía, la medicina y el derecho.

- El ejercicio formal de diseñar un modelo de simulación por computadora puede ser más valioso que la simulación real en sí. El conocimiento obtenido al diseñar un estudio de simulación sirve para cristalizar el pensamiento del analista y, a menudo, sugiere cambios en el sistema que se está simulando. Los efectos de estos cambios se pueden probar mediante simulación antes de implementarlos en el sistema real.

- La simulación puede arrojar información valiosa sobre el problema de identificar qué variables son importantes y cuáles tienen efectos insignificantes en el sistema, y puede arrojar luz sobre cómo interactúan estas variables; véase el Capítulo 7.

- La simulación se puede utilizar para experimentar con nuevos escenarios a fin de obtener información sobre el comportamiento del sistema en nuevas circunstancias.

- La simulación proporciona una *en silico* lab, permitiendo al analista descubrir un mejor control del sistema bajo estudio.

- La simulación hace posible el estudio de sistemas dinámicos en horizontes temporales reales, comprimidos o expandidos.
- La introducción de la aleatoriedad en un sistema puede ayudar a resolver muchos problemas de optimización y conteo; consulte los capítulos 6 a 10.

Como metodología de modelado, la simulación no es de ninguna manera ideal. Algunas de sus deficiencias y varias advertencias son: La simulación proporciona *estimaciones estadísticas* más bien que *exacto* características y medidas de rendimiento del modelo. Por lo tanto, los resultados de la simulación están sujetos a incertidumbre y contienen errores experimentales. Además, el modelado de simulación suele consumir mucho tiempo y, en consecuencia, es costoso en términos de tiempo del analista. Finalmente, los resultados de la simulación, sin importar cuán precisos, exactos e impresionantes, brindan información útil de manera constante sobre el sistema real. *solamente* si el modelo es una representación válida del sistema bajo estudio.

3.2.1 Clasificación de los modelos de simulación

Los modelos de simulación por computadora se pueden clasificar de varias maneras:

1. *Modelos estáticos versus dinámicos*. Los modelos estáticos son aquellos que no evolucionan con el tiempo y por tanto no representan el paso del tiempo. Por el contrario, los modelos dinámicos representan sistemas que evolucionan con el tiempo (por ejemplo, el funcionamiento de los semáforos).
2. *Modelos deterministas versus estocásticos.* Si un modelo de simulación contiene *solamente* componentes deterministas (es decir, no aleatorios), se llama *determinista*. En un modelo determinista, todas las relaciones matemáticas y lógicas entre elementos (variables) se fijan de antemano y no están sujetas a incertidumbre. Un ejemplo típico es un sistema complicado y analíticamente imposible de resolver de ecuaciones diferenciales estándar que describen, por ejemplo, una reacción química. Por el contrario, un modelo con al menos una variable de entrada aleatoria se denomina modelo. *estocástico* modelo. La mayoría de los sistemas de colas e inventarios se modelan estocásticamente.
3. *Modelos de simulación continuos versus discretos.* En los modelos de simulación discreta, la variable de estado cambia instantáneamente en puntos discretos en el tiempo, mientras que en los modelos de simulación continua, el estado cambia continuamente a lo largo del tiempo. Un modelo matemático destinado a calcular una solución numérica para un sistema de ecuaciones diferenciales es un ejemplo de simulación continua, mientras que los modelos de colas son ejemplos de simulación discreta.

Este capítulo trata de la simulación discreta y en particular de *simulación de eventos discretos* (DES) modelos. Los sistemas asociados están impulsados por la ocurrencia de eventos discretos y su estado generalmente cambia con el tiempo. Distinguiremos además entre los llamados *sistemas estáticos de eventos discretos* (DES) y *sistemas dinámicos de eventos discretos* (DEDS). La diferencia fundamental entre DESS y DEDS es que los primeros no evolucionan con el tiempo, mientras que los segundos sí. Una red de colas es un ejemplo típico de un DEDS. Un DESS generalmente implica evaluar (estimar) sumas o integrales multidimensionales complejas a través de la simulación de Monte Carlo.

Observación 3.2.1 (Cómputo en paralelo) Los recientes avances en la tecnología informática han permitido el uso de *paralela* o *repartido* simulación, donde la simulación de eventos discretos se lleva a cabo en múltiples computadoras conectadas (en red), operando simultáneamente de manera cooperativa. Dicho entorno permite la distribución simultánea de diferentes tareas informáticas entre los procesadores individuales, lo que reduce el tiempo total de simulación.

3.3 RELOJ DE SIMULACIÓN Y LISTA DE EVENTOS PARA DEDS

Recuerde que los DEDS evolucionan con el tiempo. En particular, estos sistemas cambian su estado solo en un número contable de puntos de tiempo. Los cambios de estado son desencadenados por la ejecución de eventos de simulación que ocurren en los puntos de tiempo correspondientes. Aquí, un *evento*es una colección de atributos (valores, tipos, banderas, etc.), entre los que destacan los*tiempo de ocurrencia del evento*—o simplemente*hora del evento*—la*tipo de evento*, y un algoritmo asociado para ejecutar cambios de estado.

Debido a su naturaleza dinámica, DEDS requiere un mecanismo de cronometraje para avanzar el tiempo de simulación de un evento a otro a medida que la simulación evoluciona con el tiempo. El mecanismo que registra el tiempo de simulación actual se llama *reloj de simulación*. Para realizar un seguimiento de los eventos, la simulación mantiene una lista de todos los eventos pendientes. Esta lista se llama*lista de eventos*, y su tarea es mantener todos los eventos pendientes en*cronológico*ordenar. Es decir, los eventos están ordenados por su tiempo de ocurrencia. En particular, el evento más inminente siempre se ubica al principio de la lista de eventos.

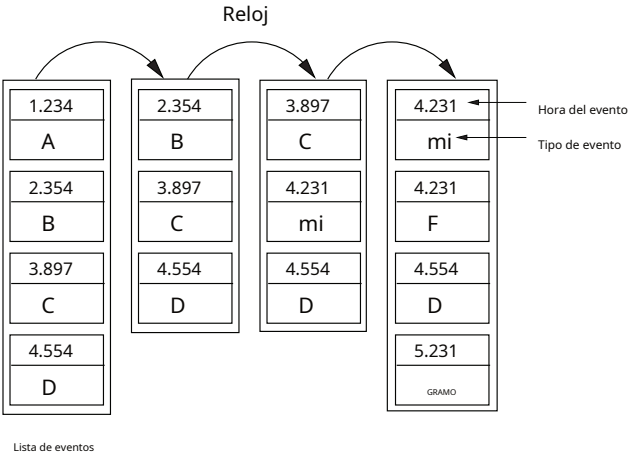

Figura 3.1: El avance del reloj de simulación y la lista de eventos.

La situación se ilustra en la figura 3.1. La simulación comienza cargando los eventos iniciales en la lista de eventos (ordenados cronológicamente), en este caso cuatro eventos. A continuación, el evento más inminente se descarga de la lista de eventos para su ejecución y el reloj de simulación avanza hasta su hora de ocurrencia, 1.234. Después de procesar y eliminar este evento, el reloj avanza al siguiente evento, que ocurre en el tiempo 2.354. En el curso de la ejecución de un evento actual, según su tipo, se actualiza el estado del sistema y posiblemente se generen y carguen (o eliminen) eventos futuros en la lista de eventos. En el ejemplo anterior, el tercer evento, de tipo C, que ocurre en el tiempo 3.897, programa un nuevo evento de tipo E en el tiempo 4.231.

El proceso de descarga de eventos de la lista de eventos, avance del reloj de simulación y ejecución del próximo evento más inminente finaliza cuando se cumple alguna condición de detención específica, por ejemplo, tan pronto como un número prescrito de clientes sale del sistema. El siguiente ejemplo lo ilustra*avance de tiempo del próximo evento* Acercarse.

EJEMPLO 3.1

El dinero ingresa a una determinada cuenta bancaria de dos maneras: a través de pagos pequeños frecuentes y pagos grandes ocasionales. Suponga que los tiempos entre pagos frecuentes subsiguientes son independientes y están uniformemente distribuidos en el intervalo continuo [7,10] (en días); y, del mismo modo, los tiempos entre pagos ocasionales posteriores son independientes y se distribuyen uniformemente el [25,35]. Cada pago frecuente se distribuye exponencialmente con una media de 16 unidades (p. ej., una unidad es $1000), mientras que los pagos ocasionales siempre tienen un tamaño de 100. Se supone que todos los intervalos y tamaños de pago son independientes. Se debita dinero de la cuenta en momentos que forman un proceso de Poisson con tasa 1 (por día), y la cantidad cargada se distribuye normalmente con media 5 y desviación estándar 1. Suponga que la cantidad inicial de dinero en la cuenta bancaria es de 150 unidades .

Tenga en cuenta que el estado del sistema, el saldo de la cuenta, cambia solo en momentos discretos. Para simular este DEDS, solo es necesario realizar un seguimiento de cuándo se producen los próximos pagos frecuentes y ocasionales, así como el próximo retiro. Denote estos tres tipos de eventos por 1, 2 y 3, respectivamente. Ahora podemos implementar la lista de eventos simplemente como un 3×2, donde cada fila contiene la hora del evento y el tipo de evento. Después de cada avance del reloj, la hora actual del evento *t*y tipo de evento *i*se registran y th Siguiente, para cada tipo de evento *i*=1,2,3, la misma t usando su distribución de intervalo correspondiente. Para e es 2, entonces se programa otro evento de tipo 2 en un *tu*~tu[0,1]. Tenga en cuenta que este evento se puede almacenar en el evento actual que se acaba de borrar. Sin embargo, se recurre entonces a poner los hechos en orden cronológico.

Una realización del proceso estocástico.{*Xt*,0 - saldo de la cuenta en el momento*t*, se da en la Figura 3.2, después de la implementación.

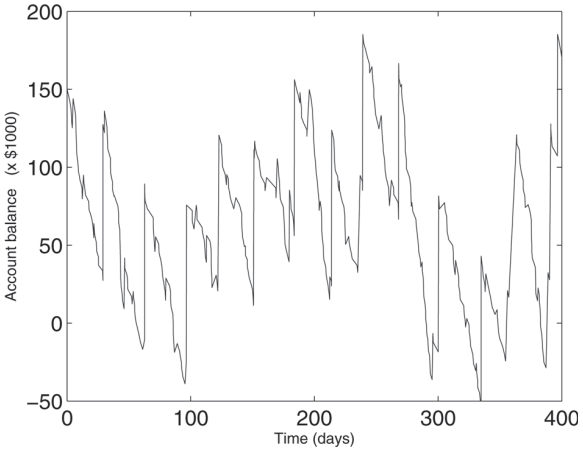

Figura 3.2: Realización del proceso de saldo de cuenta simulado.

Programa Matlab

```
limpiar todo
t = 400;
x = 150; %cantidad inicial de dinero. xx =
[150]; tt = [0];
t=0;
ev_list = inf*unos(3,2); ev_list(1,:) = [7      % tiempo de registro, escriba
+ 3*rand, 1];                                   %programar evento tipo 1
ev_list(2,:) = [25 + 10*rand,2]; %programar evento tipo 2
ev_list(3,:) = [-log(rand),3];                  %programar evento tipo 3 %
ev_list = sortrows(ev_list,1);                  ordenar lista de eventos
mientras que t < T
      t = ev_list(1,1);
      tipo_ev = lista_ev(1,2);
      cambiar ev_type
            caso 1
                  x = x + 16*-log(rand); ev_list(1,:) = [7 +
                  3*rand + t, 1]; caso 2

                  x = x + 100;
                  ev_list(1,:) = [25 + 10*rand + t, 2]; caso 3

                  x = x - (5 + randn); ev_list(1,:) = [-log(rand)
                  + t, 3];
      final
      ev_list = sortrows(ev_list,1); % ordenar lista de eventos xx
      = [xx,x];
      tt = [tt, t];
final
trama (tt, xx)
```

3.4 SIMULACIÓN DE EVENTOS DISCRETOS

Como se mencionó, DES es el marco estándar para la simulación de una gran clase de modelos en los que el sistema *estado* (una o más cantidades que describen la condición del sistema) debe observarse solo en ciertas épocas críticas (tiempos de eventos). Entre estas épocas, el estado del sistema permanece igual o cambia de manera predecible. Explicamos más las ideas detrás de DES a través de dos ejemplos más.

3.4.1 Cola en tándem

La figura 3.3 muestra un sistema de cola simple, que consta de dos colas en tándem, llamado (Jackson) *cola en tándem*. Los clientes llegan a la primera cola según un proceso de Poisson con tasa λ. El tiempo de servicio de un cliente en la primera cola se distribuye exponencialmente con tasa m_1. Los clientes que salen de la primera cola entran en la segunda. El tiempo de servicio en la segunda cola tiene una distribución exponencial con tasa m_2. Todos los tiempos entre llegadas y servicios son independientes.

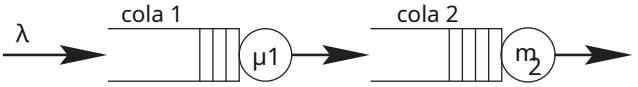

Figura 3.3: Una cola en tándem de Jackson.

Supongamos que estamos interesados en el número de clientes, X_t y Y_t, en la primera y segunda cola, respectivamente, donde consideramos a un cliente que está siendo atendido como parte de la cola. La Figura 3.4 muestra una realización típica de los procesos de longitud de cola $\{X_t, t\text{-}0\}$ y $\{Y_t, t\text{-}0\}$ obtenido a través de DES.

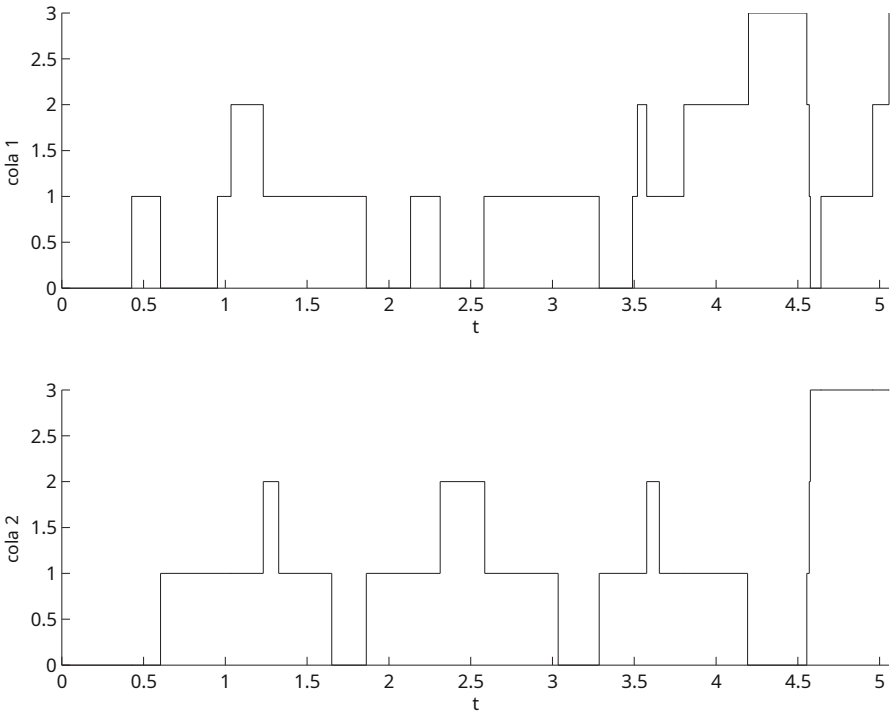

Figura 3.4: Una realización de los procesos de longitud de cola (X_t, t-0) y (Y_t, t-0).

Antes de analizar cómo simular los procesos de longitud de cola a través de DES, observe que el sistema evoluciona a través de una secuencia de eventos discretos, como se ilustra en la figura 3.5. Específicamente, el estado del sistema (X_t, Y_t) cambia solo en los momentos de llegada a la primera cola (indicada por A), salida de la primera cola (indicada por D1) y salida de la segunda cola (D2).

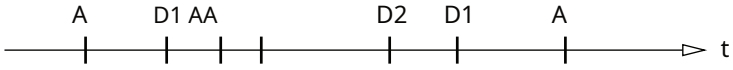

Figura 3.5: Secuencia de eventos discretos (A = llegada, D1 = salida de la primera cola, D2 = salida de la segunda cola).

Hay dos enfoques fundamentales para DES, llamados el *orientado a eventos* y *orientado al proceso* enfoques. El pseudocódigo para una implementación orientada a eventos de la cola en tándem se muestra en la figura 3.6. El programa consiste en una subrutina principal y subrutinas separadas para cada evento. Además, el programa mantiene una lista ordenada de eventos actuales y futuros programados, los llamados *lista de eventos*. Cada evento en la lista de eventos tiene un evento *escribe* ('A', 'D1' y 'D2') y un evento *tiempo* (la hora en que se producirá la llegada o la salida). La función de la subrutina principal es principalmente avanzar a través de la lista de eventos y llamar a las subrutinas que están asociadas con cada tipo de evento.

Principal

```
1  inicializar: t←0, X←0, y←0
2  Horario 'A' en t+Exp(λ).
3  mientras que cierto
4  |  Obtenga el primer evento en la lista de eventos.
5  |  Dejar t sea el momento de este evento (ahora actual).
6  |  cambiar tipo de evento actual hacer
7  |  |  caso 'A': Llamar Caso de
8  |  |  llegada 'D1': Llamar Salida1
9  |  |  caso 'D2': Llamar Salida2
10 |  Elimina el evento actual de la lista de eventos y ordena la lista de eventos.
```

Figura 3.6: Subrutina principal de un programa de simulación orientado a eventos.

La función de las subrutinas de eventos es actualizar el estado del sistema y programar nuevos eventos en la lista de eventos. Por ejemplo, un evento de llegada en el momento t activará otro evento de llegada a la vez $t+Z$, con $Z \sim \text{Exp}(\lambda)$. Escribimos esto, como en la rutina principal, en forma abreviada como $t+\text{Exp}(\lambda)$. Además, si la primera cola está vacía, también activará un evento de salida de la primera cola en el momento $t+\text{Exp}(m_1)$.

Llegada

```
1  Planificar una'
     a t+Exp(λ).
2  si X=0 después
3  |  Horario 'D1'
   |    a t+Exp(m1).
4  X←X+1
```

Salida1

```
1  X←x −1
2  si X=0 después
3  |  Horario 'D1'
   |    a t+Exp(m1).
4  si y=0 después
5  |  Horario 'D2'
   |    a t+Exp(m2).
6  y←y+1
```

Salida2

```
1  y←y -1
2  si y=0 después
3  |  Horario 'D2'
   |    a t+Exp(m2).
```

Figura 3.7: Subrutinas de eventos de un programa de simulación orientado a eventos.

El enfoque de DES orientado a procesos es mucho más flexible que el enfoque orientado a eventos. Un programa de simulación orientado a procesos se parece mucho al

procesos reales que impulsan la simulación. Dichos programas de simulación se escriben invariablemente en un lenguaje de programación orientado a objetos, como Java o C++. Ilustramos el enfoque orientado al proceso a través de nuestro ejemplo de cola en tándem. En contraste con el enfoque orientado a eventos, los clientes, servidores y colas ahora son entidades reales, o *objetos* en el programa, que puede ser manipulado. Las colas son objetos pasivos que pueden contener varios clientes (o estar vacíos), y los propios clientes pueden contener información como sus horas de llegada y salida. Los servidores, sin embargo, son objetos activos (*procesos*) que pueden interactuar entre sí y con los objetos pasivos. Por ejemplo, el primer servidor toma un cliente de la primera cola, atiende al cliente y lo coloca en la segunda cola cuando termina, alertando al segundo servidor que ha llegado un nuevo cliente si es necesario. Para generar las llegadas, definimos un *generador* proceso que genera un cliente, lo coloca en la primera cola, alerta al primer servidor si es necesario, se mantiene durante un tiempo aleatorio entre llegadas (suponemos que los tiempos entre llegadas son iid) y luego repite estas acciones para generar el siguiente cliente.

Como en el enfoque orientado a eventos, existe una lista de eventos que realiza un seguimiento de los eventos actuales y pendientes. Sin embargo, esta lista de eventos ahora contiene *procesos*. El proceso en la parte superior de la lista de eventos es el que está actualmente activo. Los procesos pueden ACTIVAR otros procesos colocándolos al principio de la lista de eventos. Los procesos activos pueden MANTENER su acción durante un cierto período de tiempo (dichos procesos se colocan más arriba en la lista de eventos). Los procesos pueden PASIVARSE por completo (eliminarse temporalmente de la lista de eventos). La figura 3.8 enumera la estructura típica de un programa de simulación orientado a procesos para la cola en tándem.

Principal

1 **inicializar:** cree las dos colas, los dos servidores y el generador.
2 ACTIVAR el generador.
3 HOLD (duración de la simulación).
4 DETÉNGASE

Generador

1 **mientras que cierto**
2 | Generar nuevo cliente.
3 | Ponga al cliente en la primera fila.
4 **si** el servidor 1 está inactivo **después**
5 | ACTIVAR servidor 1.
6 HOLD (tiempo entre llegadas).

Servidor 1

1 **mientras que cierto**
2 | **si** *la sala de espera está vacía* **después**
3 | | PASIVADO
4 | **más**
5 | | Obtenga el primer cliente de la sala de espera.
6 | | HOLD (tiempo de servicio).
7 | Poner al cliente en la cola 2. **si** el
8 | servidor 2 está inactivo **después**
9 | | ACTIVAR servidor 2.

Figura 3.8: La estructura de un programa de simulación orientado a procesos para la cola en tándem. El proceso del servidor 2 es similar al proceso del servidor 1, con las líneas 7 a 9 reemplazadas por "eliminar cliente del sistema".

La recopilación de estadísticas (por ejemplo, los tiempos de espera o las longitudes de las colas) puede ser realizada por diferentes objetos y en varias etapas de la simulación. Por ejemplo, los clientes pueden registrar sus horas de llegada y salida e informarlas o registrarlas justo antes de que abandonen el sistema. Hay muchos entornos de simulación orientados a objetos disponibles gratuitamente en la actualidad, como SSJ, SimPy y C++Sim, todos inspirados en el lenguaje de simulación pionero SIMULA.

3.4.2 Problema del reparador

Imaginar *norte* máquinas trabajando simultáneamente. Las máquinas no son confiables y fallan de vez en cuando. Existen *metro* < *norte* reparadores idénticos que pueden trabajar cada uno solo en una máquina a la vez. Cuando una máquina ha sido reparada, está como nueva. Cada máquina tiene una distribución de tiempo de vida y una distribución de tiempo de reparación fijas. Suponemos que la vida útil y los tiempos de reparación son independientes entre sí. Dado que hay menos reparadores que máquinas, puede ocurrir que una máquina falle y todos los reparadores estén ocupados reparando otras máquinas averiadas. En ese caso, la máquina averiada se coloca en una cola para que la atienda el siguiente técnico disponible. Cuando al finalizar un trabajo de reparación, un reparador encuentra que la cola de la máquina fallida está vacía, ingresa al grupo de reparación y permanece inactivo hasta que se requiere su servicio nuevamente. Suponemos que las máquinas y los reparadores ingresan a sus respectivas colas en forma de primero en entrar, primero en salir (FIFO). El sistema se ilustra en la Figura 3.

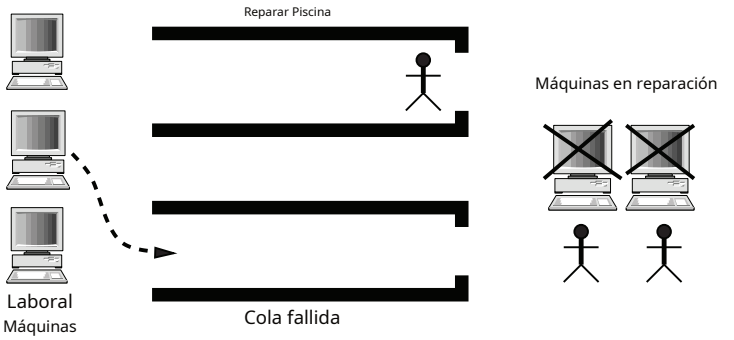

Figura 3.9: El sistema de reparación.

Para este modelo en particular, el estado del sistema podría estar compuesto por el número de reparadores disponibles. *R* y el número de máquinas fallidas *F* en cualquier momento *t*. En general, el proceso estocástico $\{(F_t, R_t), t\text{-}0\}$ no es un proceso de Markov a menos que el servicio y la duración tengan distribuciones exponenciales.

Al igual que con la cola en tándem, primero describimos un enfoque orientado a eventos y luego un enfoque orientado a procesos para este modelo.

3.4.2.1 Enfoque orientado a eventos Hay dos tipos de eventos: eventos de falla 'F' y eventos de reparación 'R'. Cada evento desencadena la ejecución del correspondiente procedimiento de falla o reparación. La tarea del programa principal es hacer avanzar el reloj de simulación y asignar el procedimiento correcto a cada evento. denotar por *norte* el número de máquinas fallidas y por *norte* el número de reparadores disponibles, escribimos el programa principal de la siguiente forma:

Programa principal

```
inicializar: Dejar t←0, norte_r←metro y norte_F←0. por i=1 a norte. hacer
    Horario 'F' de la máquina i en el momento t+vida(i).
mientras que cierto
    Obtenga el primer evento en la lista de eventos.
    Dejar t sea el momento de este evento (ahora actual).
    Dejar i ser el número de máquina asociado con este evento.
    cambiar tipo de evento actual hacer
        caso 'F': Llamar Caso de
        falla 'R': Llamar Reparar
    Elimina el evento actual de la lista de eventos.
```

En caso de falla, se debe programar una reparación a una hora igual a la hora actual más el tiempo de reparación requerido para la máquina en particular. Sin embargo, esto es cierto solo si hay un reparador disponible para realizar las reparaciones. Si este no es el caso, la máquina se coloca en la cola de "fallidos". El número de máquinas fallidas siempre se incrementa en 1. El procedimiento de falla es el siguiente:

Procedimiento de falla

```
si norte_r>0 después
    Horario 'R' de la máquina i en el momento t+tiempo de reparación(i).
    norte_r←norte_r−1
más
    Agregue la máquina a la cola de reparación.
norte_F←norte_F+1
```

Tras la reparación, el número de máquinas con fallas se reduce en 1. La máquina que acaba de repararse está programada para su próxima falla. Si la cola de "fallidos" no está vacía, el reparador toma la siguiente máquina de la cola y programa el evento de reparación correspondiente. De lo contrario, el número de reparadores inactivos/disponibles aumenta en 1. Esto da el siguiente procedimiento de reparación:

Procedimiento de reparación

```
norte_F←norte_F−1
Horario 'F' para máquina i en el momento t+vida(i).
si piscina de reparación no vacía después
    Retire la primera máquina de la cola "fallida"; dejar j sea su número. Horario 'R' de
    la máquina j en el momento t+tiempo de reparación(j).
más
    norte_r←norte_r+1
```

3.4.2.2 Enfoque orientado a procesos Para delinear un enfoque orientado a procesos para cualquier simulación, es conveniente representar los procesos mediante diagramas de flujo. En este caso hay dos procesos: el proceso del reparador y el proceso de la máquina. Los diagramas de flujo de la figura 3.10 se explican por sí mismos. Tenga en cuenta que las líneas paralelas horizontales en los diagramas de flujo indican que el proceso PASIVA, es decir, el proceso

se detiene temporalmente (se elimina de la lista de eventos), hasta que otro proceso lo ACTIVA. Las letras A y B dentro de un círculo indican cómo interactúan los dos. Una cruz en el diagrama de flujo indica que el proceso está reprogramado en la lista de eventos (EL). Esto sucede en particular cuando el proceso se DETIENE durante un período de tiempo. Después de sostener, se reanuda desde donde lo dejó.

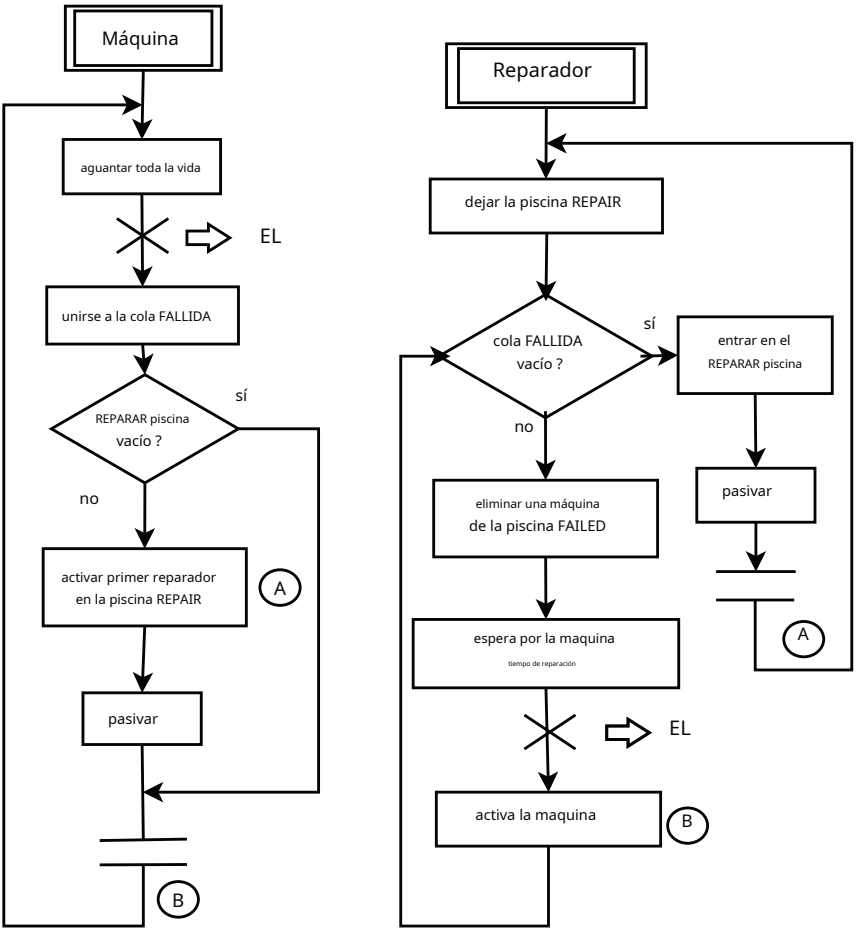

Figura 3.10: Diagramas de flujo para los dos procesos en el problema del reparador.

PROBLEMAS

3.1 Considera el $M/M/1$ sistema de colas en el Ejemplo 1.13. Dejar X_t sea el número de clientes en el sistema en ese momento t. Escribir un programa de computadora para simular el proceso estocástico $X=\{X_t,\ t=0\}$ utilizando un enfoque DES orientado a eventos o procesos. Presentar rutas de muestra del proceso para los casos. $\lambda = 1$, $m=2$ y $\lambda=10$, $m=11$

3.2 Repita la simulación anterior, pero ahora supongatu(0,2) tiempos entre llegadas y tu(0,1⁄2) tiempos de servicio (todos independientes).

3.3 Ejecute el programa Matlab del ejemplo 3.1 (o impleméntelo en el lenguaje informático que prefiera). De 1000 ejecuciones, ¿cuántas conducen a un saldo de cuenta negativo durante los primeros 100 días? ¿Cómo se comporta el proceso para grandes$t$?

3.4 Implemente un programa de simulación orientado a eventos para la cola en tándem. Deje que las llegadas entre llegadas se distribuyan exponencialmente con media 5, y que los tiempos de servicio se distribuyan uniformemente en [3,6]. Trazar realizaciones de los procesos de longitud de cola de ambas colas.

3.5 Considere el problema del reparador con dos máquinas idénticas y un reparador. Suponemos que la vida útil de una máquina tiene una distribución exponencial con expectativa 5 y que el tiempo de reparación de una máquina es exponencial con expectativa 1. Todas las vidas útiles y los tiempos de reparación son independientes entre sí. Dejar X_tSea el número de máquinas fallidas a la vez$t$.

a)Comprueba eso$X=\{X_t,\ t\text{-}0\}$es un proceso de nacimiento y muerte, y dar las correspondientes tasas de nacimiento y muerte.

b)Escriba un programa que simule el proceso.Xde acuerdo con el Algoritmo 2.7.2 y utilice esto para evaluar la fracción de tiempo que ambas máquinas están fuera de servicio. Simular desde$t$=0 at=100.000.

C)Escriba un programa de simulación orientado a eventos para este proceso.

d)Deje que la vida exponencial y los tiempos de reparación se distribuyan uniformemente, en [0,10] y [0,2], respectivamente (por lo tanto, las expectativas siguen siendo las mismas que antes). Simular desde$t$=0 at=100.000. ¿Cómo cambia la fracción de tiempo que ambas máquinas están fuera de servicio?

mi) Ahora simule un problema de reparador con la vida y los tiempos de reparación indicados anteriormente, pero ahora con cinco máquinas y tres reparadores. Ejecutar de nuevo desde t=0 at=100.000.

3.6 Dibuje diagramas de flujo, como en la Figura 3.10, para todos los procesos en la cola tándem; véase también la figura 3.8.

3.7 Considere el siguiente sistema de colas. Los clientes llegan a un círculo, según un proceso de Poisson con tasaλ. En el círculo, que tiene una circunferencia 1, un solo servidor viaja a velocidad constanteα^{-1}. Al llegar, los clientes eligen sus posiciones en el círculo de acuerdo con una distribución uniforme. El servidor siempre se mueve hacia el cliente más cercano, a veces en el sentido de las agujas del reloj, a veces en el sentido contrario a las agujas del reloj. Al llegar a un cliente, el servidor se detiene y atiende al cliente de acuerdo con una distribución de tiempo de servicio exponencial con parámetro$m$. Cuando el servidor finaliza, el cliente se elimina del círculo y el servidor reanuda su viaje en el círculo. Dejar$\eta=\lambda\alpha$, y deja$X_t\in[0,1]$ sea la posición del servidor en el momento t. Además, deja$norte_t$ sea el número de clientes que esperan en el círculo a la vez$t$. Implementar un programa de simulación para este llamado*sistema de poling continuo con un servidor "codicioso"*, y trazar realizaciones de los procesos$\{X_t,\ t\text{-}0\}$y$\{norte_t,\ t\text{-}0\}$, tomando los parámetrosλ= 1,m= 2, para diferentes valores deα. Tenga en cuenta que aunque el espacio de estado de$\{X_t,\ t\text{-}0\}$es continuo, el sistema sigue siendo un DEDS, ya que entre los eventos de llegada y servicio el estado del sistema cambia de forma determinista.

3.8 Considere un*línea de flujo continuo*que consta de tres máquinas en tándem separadas por dos áreas de almacenamiento, o amortiguadores, a través de las cuales fluye un flujo continuo (fluido) de artículos de una máquina a la siguiente; consulte la figura 3.11.

Figura 3.11: Una línea de flujo con tres máquinas y dos amortiguadores (línea de flujo de tres etapas).

cada maquina$i$=1,2,3 tiene un específico*velocidad de la máquina* v_i, que es la velocidad máxima a la que puede transferir productos desde su búfer ascendente a su búfer descendente. La vida útil de la máquina$i$tiene una distribución exponencial con parámetroλ_i. La reparación de la máquina$i$comienza inmediatamente después de la falla y requiere un tiempo exponencial con parámetrom_i. Se supone que todos los tiempos de vida y reparación son independientes entre sí. Las fallas son independientes de la operación. En particular, la tasa de fallas de una máquina "hambrienta" (una máquina que está inactiva porque no recibe información de su búfer ascendente) es la misma que la de una máquina completamente operativa. La primera máquina tiene un suministro ilimitado.

Suponga que todas las velocidades de la máquina son 1, los búferes son del mismo tamañob, y todas las máquinas son idénticas con los parámetrosλ= 1 ym= 2

a)Implemente un programa de simulación orientado a eventos o procesos para este sistema.

b)Evaluar mediante simulación el promedio*rendimiento*del sistema (la cantidad de fluido a largo plazo que entra/sale del sistema por unidad de tiempo) en función del tamaño del buffer$b$.

Otras lecturas

Uno de los primeros libros sobre simulación Monte Carlo es de Hammersley y Handscomb [3]. Kalos y Whitlock [4] es otra referencia clásica. Los enfoques orientados a eventos y procesos para la simulación de eventos discretos se explican elegantemente en Mitrani [6]. Entre la gran variedad de libros sobre DES, todos centrados en diferentes aspectos del proceso de modelado y simulación, mencionamos [5], [8], [1] y [2]. La elección del lenguaje informático en el que implementar un programa de simulación es muy subjetiva. Los modelos simples discutidos en este capítulo pueden implementarse en cualquier lenguaje de programación estándar, incluso Matlab, aunque este último no proporciona una manipulación fácil de la lista de eventos. Los entornos de simulación comercial como ARENA/SIMAN y SIMSCRIPT II.5 facilitan mucho la implementación de modelos más grandes. Alternativamente, Existen varios paquetes gratuitos de Java similares a SIMULA que ofrecen una implementación rápida de programas de simulación orientados a eventos y procesos. Ejemplos son el SSJ de Pierre L'Ecuyerhttp://www.iro.umontreal.ca/~simardr/ssj/,DSOL http://sk-3.tbm.tudelft.nl/simulación/,desarrollado por la Universidad Técnica de Delft, y el SimPy basado en pythonhttps://pypi.python.org/pypi/simpy.