Capítulo 6

Generación de variables aleatorias continuas

6.1. Introducción

Se dice que una variable aleatoria X es continua, o más propiamente, absolutamente continua, si existe $f : \mathbb{R} \mapsto \mathbb{R}$ tal que

$$F(x) := P(X \leq x) = \int_{-\infty}^{x} f(t)\, dt.$$

Al igual que para las variables aleatorias discretas, analizaremos dos métodos de generación para variables continuas, con las respectivas mejoras en cada caso particular. Estos métodos también se denominan:

- Método de la transformada inversa.
- Método de aceptación y rechazo.

6.2. Método de la transformada inversa

La generación o simulación de valores de una variable aleatoria por el método de la transformada inversa requiere conocer la función de distribución acumulada de la variable X a simular. Si X es (absolutamente) continua y f es su función de densidad, la función de distribución acumulada:

$$F(x) = \int_{-\infty}^{x} f(t)\, dt,$$

cumple la propiedad de ser continua, no decreciente, y:

$$\lim_{x \to -\infty} F(x) = 0, \qquad \lim_{x \to \infty} F(x) = 1.$$

En esta sección nos referiremos a variables aleatorias X cuya función de distribución sea monótona creciente sobre el conjunto $F^{-1}(0,1) = \{x \mid 0 < F(x) < 1\}$. De esta manera, la función F tiene inversa sobre el intervalo $(0,1)$. Para los casos en que no se cumple esta propiedad también es posible definir métodos de simulación de las variables, pero las demostraciones de su validez son un tanto más complejas.

Proposición 6.1. Sea $U \sim \mathcal{U}(0,1)$ una variable aleatoria. Para cualquier función de distribución continua F, monótona creciente en $F^{-1}(0,1)$, la variable aleatoria X definida por:

$$X = F^{-1}(U)$$

tiene distribución F.

Notemos que U toma valores en el intervalo $(0,1)$, excluyendo los puntos 0 y 1. La propiedad de monotonía de F asegura que F tiene inversa en $(0,1)$. Así, si $X = F^{-1}(U)$, su función de distribución está dada por:

$$F_X(x) = P(X \leq x) = P(F^{-1}(U) \leq x).$$

Como F es creciente, entonces $a \leq b$ si y sólo si $F(a) \leq F(b)$. En particular, los conjuntos:

$$\{F^{-1}(U) \leq x\} \qquad \text{y} \qquad \{U \leq F(x)\}$$

son iguales. Así resulta:

$$F_X(x) = P(F(F^{-1}(U)) \leq F(x)) = P(U \leq F(x)) = F(x).$$

Luego X tiene distribución F.

Esta proposición da un método en principio simple para simular una variable aletoria continua con función de distribución acumulada F:

```
def Tinversa():
    U=random()
    return G(U)  # G = F^{-1}
```

Sin embargo, pueden existir algunas complicaciones. Por ejemplo, que la inversa de F involucre funciones computacionalmente costosas ($\sqrt[n]{f(x)}$, $\log(x)$, etc.), o que no pueda ser calculada explícitamente (por ejemplo la distribución de una variable normal, o de una gamma).

Aparecen entonces otras estrategias para generar estas variables, como por ejemplo expresando a F como la distribución del mínimo o del máximo de variables aleatorias independientes, o de la suma de variables independientes, o distribuciones condicionales u otros casos particulares.

Ejemplo 6.1. Escribir un método para generar el valor de una variable aleatoria X con función de densidad

$$f(x) = -\frac{x}{2} + 1, \qquad 0 \leq x \leq 2.$$

$$F(x) = \begin{cases} 0 & x \leq 0 \\ -\frac{x^2}{4} + x & 0 < x < 2 \\ 1 & x \geq 2 \end{cases}$$

F resulta entonces monótona creciente en el intervalo $(0, 2)$.

Para simular X debemos determinar la inversa de F en $(0, 1)$. Es decir, para cada $u \in (0, 1)$, encontrar el valor x tal que

$$-\frac{x^2}{4} + x = u.$$

Esta ecuación tiene dos soluciones para x:

$$\begin{cases} x = 2 + 2\sqrt{1-u} \\ \\ x = 2 - 2\sqrt{1-u} \end{cases} \quad \text{o}$$

pero sólo la segunda pertenece al intervalo $(0, 2)$. Luego el algoritmo de simulación de esta variable por el método de la transformada inversa es:

```
def inversaX():
    U = 1 - random()
    return 2 - 2 * sqrt(U)
```

También, utilizando que U y $1 - U$ están igualmente distribuidas, vale el algoritmo:

```
def inversaX():
    U = random()
    return 2 - 2 * sqrt(U)
```

Ejemplo 6.2. Escribir un método para simular una variable aleatoria Y con función de densidad

$$f(x) = \begin{cases} 0.25 & 0 \leq x \leq 2 \\ 0.5 & 2 < x < 3 \\ 0 & \text{en otro caso.} \end{cases}$$

Para esta variable aleatoria la función de distribución acumulada está dada por:

$$F(x) = \begin{cases} 0 & x \leq 0 \\ \frac{x}{4} & 0 \leq x \leq 2 \\ \frac{x-1}{2} & 2 < x < 3 \\ 1 & x \geq 3 \end{cases}$$

Notemos que Y toma valores en el intervalo $(0, 3)$, y que $F(2) = 0.5$. Luego el algoritmo de simulación de la variable es:

```
def inversaY():
    U = random()
    if U<0.5:
        return 4*U
    else:
        return 2*U+1
```

Ejemplo 6.3. Consideremos $X_1, X_2, \ldots, X_n$ n variables aleatorias independientes, con funciones de distribución acumulada $F_1, F_2, \ldots, F_n$, respectivamente:

$$F_i(a) = P(X_i \leq a), \qquad i = 1, 2, \ldots, n.$$

Llamamos X a la variable aleatoria que es el máximo de las X_i, $1 \leq i \leq n$:

$$X = \text{máx}\{X_1, X_2, \ldots, X_n\}.$$

Entonces la función de distribución acumulada de X está dada por:

$$F_X(a) = F_1(a) \cdot F_2(a) \ldots F_n(a).$$

En particular, si X_1, X_2, …, X_n son uniformes en $(0, 1)$, independientes entre sí, tenemos que $F_i(x) = x$, $0 \leq x \leq 1$ e $1 \leq i \leq n$. Luego el máximo de las X_i tiene una distribución dada por:

$$F_X(x) = x^n, \qquad 0 \leq x \leq 1.$$

Así, si se desea simular una variable aleatoria con distribución $F_X(x) = x^n$, en lugar de tomar una raíz n-ésima también es posible generar n uniformes y tomar su máximo. ¿Es más eficiente?

```
def raizn(n):
    Maximo = 0.
    for _ in range(n):
        Maximo = max(Maximo,random())
    return Maximo
```

6.2.1. Simulación de una variable aleatoria exponencial

Si X es una variable aleatoria con distribución exponencial con parámetro $\lambda = 1$, $X \sim \mathcal{E}(1)$, entonces su función de distribución acumulada está dada por:

$$F(x) = \begin{cases} 1 - e^{-x} & x > 0 \\ 0 & x \leq 0. \end{cases}$$

Luego la inversa de F sobre $(0, 1)$ está dada por:

$$F^{-1}(u) = -\ln(1 - u), \qquad u \in (0, 1).$$

Así, el algoritmo de simulación para $X \sim \mathcal{E}(1)$ es:

```
def exponencial():
    U = 1-random()
    return -log(1-U)
```

Notemos además que si X es exponencial con parámetro 1, esto es, $X \sim \mathcal{E}(1)$, entonces $Y = \frac{1}{\lambda}X$ es exponencial con media $\frac{1}{\lambda}$: $Y \sim \mathcal{E}(\lambda)$. Luego el método por transformada inversa para simular valores de $Y \sim \mathcal{E}(\lambda)$ es:

```
def exponencial(lamda):
    U = 1-random()
    return -log(U)/lamda
```

6.2.2. Simulación de una variable aleatoria Poisson $X \sim \mathcal{P}(\lambda)$

Sea $N(t)$, $t \geq 0$, un proceso de Poisson homogéneo con intensidad λ. Entonces se cumple que:

- Los tiempos de llegada entre eventos son variables aleatorias exponenciales de parámetro λ, es decir, media $\frac{1}{\lambda}$.
- El número de eventos en un intervalo de tiempo de longitud t es una variable aleatoria Poisson de media $\lambda \cdot t$.

En particular $N(1)$ es una variable aleatoria Poisson de media λ que indica el número de arribos hasta el tiempo $t = 1$, y los tiempos entre arribos en el intervalo $[0, 1]$ son exponenciales de parámetro λ.

Por lo tanto, si se simulan variables aleatorias exponenciales X_1, X_2, ..., con $X_i \sim \mathcal{E}(\lambda)$ para $i \geq 1$, hasta que $X_1 + X_2 + \cdots + X_n \leq 1$ y $X_1 + X_2 + \cdots + X_n + X_{n+1} > 1$, entonces n representa el número de arribos hasta $t = 1$. Esto es:

$$N(1) = \text{máx}\{n \mid X_1 + X_2 + \cdots + X_n \leq 1\}$$

Si se simula cada exponencial X_i con $-\frac{1}{\lambda}\ln(1-U_i)$, con $U_i \sim \mathcal{U}(0,1)$, tenemos que

$$\begin{aligned} N(1) &= \text{máx}\{n \mid X_1 + X_2 + \cdots + X_n \leq 1\} \\ &= \text{máx}\{n \mid -\frac{1}{\lambda}\left(\ln(1-U_1) + \ln(1-U_2) + \cdots + \ln(1-U_n)\right) \leq 1\} \\ &= \text{máx}\{n \mid -\frac{1}{\lambda}\left(\ln((1-U_1)\cdot(1-U_2)\cdots(1-U_n)\right) \leq 1\} \\ &= \text{máx}\{n \mid \ln\left((1-U_1)\cdot(1-U_2)\cdots(1-U_n)\right) \geq -\lambda\} \\ &= \text{máx}\{n \mid (1-U_1)\cdot(1-U_2)\cdots(1-U_n) \geq e^{-\lambda}\} \end{aligned}$$

Luego:

$$N(1) = \text{mín}\{n \mid (1-U_1)\cdot(1-U_2)\cdots(1-U_n) < e^{-\lambda}\} - 1$$

```
def Poisson_con_exp(lamda):
    X = 0
    Producto = 1- random()
    cota = exp(-lamda)
    while Producto >= cota:
        Producto *= 1 - random()
        X += 1
    return X
```

6.2.3. Simulación de una variable con distribución Gamma(n,λ^{-1})

Hemos visto que la suma de n variables aleatorias exponenciales, independientes, con parámetro λ, es una variable aleatoria con distribución $Gamma(n, \lambda^{-1})$. Esta propiedad nos permite dar un algoritmo de simulación de una variable Gamma a través de la generación de n exponenciales independientes con el mismo parámetro. Notemos que si U_1, U_2, ..., U_n son valores de una variable uniforme $U \sim \mathcal{U}(0,1)$, entonces:

$$\begin{aligned} X &= -\frac{1}{\lambda}\ln(1-U_1) - \frac{1}{\lambda}\ln(1-U_2)\cdots - \frac{1}{\lambda}\ln(1-U_n) \\ &= -\frac{1}{\lambda}\ln\left((1-U_1)\cdot(1-U_2)\cdots(1-U_n)\right) \end{aligned}$$

De esta forma, un algoritmo para la simulación de una variable $Gamma(n, \lambda^{-1})$ es como sigue:

```
def Gamma(n,lamda):
    'Simula una gamma con parámetros n y 1/lamda'
    U = 1
    for _ in range(n):
        U *= 1-random()
    return -log(U)/lamda
```

Notemos que este método requiere la simulación de n variables uniformes y calcula un único logaritmo, mientras que para generar n exponenciales es necesario calcular n logaritmos. Veamos cómo puede utilizarse la generación de una $Gamma(n, \lambda^{-1})$ para generar n exponenciales independientes de parámetro λ y restringiendo el número de logaritmos a calcular.

Teorema 6.1. Si $X, Y \sim \mathcal{E}(\lambda)$, independientes, entonces

$$f_{X|X+Y}(x \mid t) = \frac{1}{t}\mathbb{I}_{(0,t)}(x),$$

es decir, X condicional a $X + Y = t$ es uniforme en $(0, t)$.

Demostración. Notemos que la función de densidad conjunta de X y $X + Y$ cumple:

$$\begin{aligned} f_{X|X+Y}(x \mid t) &= \frac{f_{X,X+Y}(x,t)}{f_{X+Y}(t)} = \frac{f_{X+Y|X}(t,x)f_X(x)}{f_{X+Y}(t)} = \frac{f_Y(t-x)f_X(x)}{f_{X+Y}(t)} \\ &= \frac{e^{-\lambda(t-x)}e^{-\lambda x}}{e^{-\lambda t}}\frac{1}{t} \cdot \mathbb{I}_{[0,t]}(x) = \frac{1}{t} \cdot \mathbb{I}_{[0,t]}(x). \end{aligned}$$

La demostración de este teorema y del caso general para n exponenciales está disponible en el material complementario. □

Luego, para generar X, Y exponenciales independientes, de parámetro λ podemos aplicar el siguiente algoritmo:

```
def DosExp(lamda):
    V1, V2 = 1-random(), 1-random()
    t = -log(V1 * V2) / lamda
    U = random()
    X = t * U
    Y = t - X
    return X, Y
```

Este algoritmo requiere el cálculo de un único logaritmo y la generación de tres uniformes.

En el caso general, para simular n exponenciales a partir de una $X \sim Gamma(n, \lambda^{-1})$ se requiere calcular un único logaritmo y $n-1$ uniformes adicionales: $V_1, V_2, \ldots, V_{n-1}, V_j \sim U(0,1)$. En primer lugar se ordenan los valores de las V_i:

$$V_{i_1} < V_{i_2} < \ldots < V_{i_{n-1}}.$$

Si $X = t$, entonces el intervalo $(0, t)$ se divide en n subintervalos consecutivos, no superpuestos, de longitud:

$$tV_{i_1}, \qquad t\,(V_{i_2} - V_{i_1}), \qquad t\,(V_{i_{n-1}} - V_{i_{n-2}}), \quad t(1 - V_{i_{n-1}}).$$

Las longitudes de cada uno de estos intervalos son variables aleatorias con distribución exponencial $\mathcal{E}(\lambda)$ e independientes entre sí. El algoritmo resulta entonces:

```
def Nexponenciales(n,lamda):
    t = 1
    for _ in range(n): t *= random()
    t = -log(t)/lamda
    unif = random.uniform(0,1,n-1)
    unif.sort()
    exponenciales = [unif[0]*t]
    for i in range(n-2):
        exponenciales.append((unif[i+1]-unif[i])*t)
    exponenciales.append((1-unif[n-2])*t)
    return exponenciales
```

6.3. El método de aceptación y rechazo

Supongamos que se quiere generar una variable aleatoria X con función de densidad f:

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)\,dt,$$

y que se tiene un método para generar otra variable Y, con densidad g, tal que

$$\frac{f(y)}{g(y)} \leq c, \qquad \text{para todo } y \in \mathbb{R} \text{ tal que } f(y) \neq 0.$$

El **método de rechazo** para generar X a partir de Y tiene el siguiente algoritmo:

```
def Aceptacion_Rechazo_X():
    while 1:
```

```
Simular Y
U = random()
if U < f(Y) / (c * g(Y)):
    return Y
```

De manera análoga al caso del método para simular variables aleatorias discretas, se puede probar que:

a) La variable aleatoria generada por el algoritmo Aceptacion_Rechazo_X() tiene densidad f.

b) El número de iteraciones del algoritmo es una variable aleatoria geométrica con media c.

Entonces, para cada variable X que se desee generar rechazando contra una variable Y, es necesario determinar una cota c para el cociente $\frac{f(x)}{g(x)}$, válida para todo $x \in \mathbb{R}$. Para determinar este valor es útil recordar algunos resultados del Análisis Matemático.

En primer lugar, considerar la función

$$h(x) = \frac{f(x)}{g(x)}, \qquad x \text{ tal que } f(x) \neq 0,$$

y para esta función determinar:

a) Puntos críticos: Esto es, $f'(x) = 0$ o $f'(x)$ no existe.

b) Analizar cuáles de estos puntos corresponden a máximos locales.

c) Evaluar h en los extremos de su dominio, si es acotado, o los límites correspondientes.

d) Elegir una cota a partir de los valores obtenidos en (b) y (c).

Ejemplo 6.4. Se quiere generar una variable aleatoria X con función de densidad:

$$f(x) = 20x(1-x)^3, \qquad 0 < x < 1.$$

En particular, X tiene distribución $Beta(2, 4)$:

$$\boxed{f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}\mathbb{I}_{(0,1)}(x) \qquad \text{Variable } Beta(\alpha, \beta)}$$

Dado que $f(x) \neq 0$ en el intervalo $(0, 1)$, se podría utilizar el método de rechazo generando una variable $Y \sim \mathcal{U}(0, 1)$. Tenemos entonces que

$$f_X(x) = 20x(1-x)^3, \qquad f_Y(x) = 1, \qquad x \in (0, 1),$$

y $h(x) = \frac{f_X(x)}{f_Y(x)} = f_X(x)$.

Los pasos para analizar una cota de h son los siguientes:

$$\begin{aligned} h(x) &= 20x(1-x)^3, \qquad 0 < x < 1 \\ h'(x) &= 20(1-x)^2 \cdot (1-4x) \end{aligned}$$

- Puntos críticos en $(0,1)$: $x = 1/4$.
- $x = 1/4$ es el único punto crítico, luego es un máximo de h.
- $h(1/4) = f(1/4) = 135/64$ es el valor máximo de h

$$c = \frac{135}{64} = 2.109375.$$

Además,

$$\frac{f(x)}{c \cdot g(x)} = 20 \cdot \frac{64}{135} x(1-x)^3 \sim 9.481481 \cdot x(1-x)^3.$$

Luego, el algoritmo para generar X rechazando con una distribución uniforme es:

```
def Beta_2_4():
    while 1:
        Y = random()
        U = random()
        if U < 9.481481 * Y * (1-Y) ** 3:
            return Y
```

Este método requiere un número promedio de ciclos del orden de $c = \dfrac{135}{64} \approx 2.11$.

Ejemplo 6.5. Analicemos cómo simular una variable aleatoria X, con densidad $Gamma(\frac{3}{2}, 1)$:

$$f(x) = \begin{cases} Kx^{1/2}e^{-x} & x > 0 \\ 0 & c.c. \end{cases}, \qquad K = \frac{1}{\Gamma(\frac{3}{2})} = \frac{2}{\sqrt{\pi}}.$$

En general, si X es una Gamma con función de densidad dada por

$$g(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} e^{-\beta x} x^{\alpha-1}, \qquad x > 0,$$

entonces su valor esperado es

$$E[X] = \frac{\alpha}{\beta}.$$

En este caso, $\alpha = \frac{3}{2}$ y $\beta = 1$, por lo cual $E[X] = \frac{3}{2}$. Dado que una variable exponencial tiene el mismo rango que X, una posibilidad entonces es utilizar el método de rechazo con una variable exponencial $Y \sim \mathcal{E}(\lambda)$ que tenga igual media que X, es decir, $Y \sim \mathcal{E}(\frac{2}{3})$.

El método de rechazo implica calcular la constante c tal que $\frac{f_X(x)}{f_Y(x)} \leq c$, si $x > 0$. Esto es, debemos hallar una cota superior para la función:

$$h(x) = \frac{Kx^{1/2}e^{-x}}{\frac{2}{3}e^{-\frac{2}{3}x}} = K\,\frac{3}{2}\,x^{1/2}e^{-\frac{1}{3}x}, \qquad x > 0.$$

Tenemos que $h'(x) = 0$ si y sólo si

$$(\frac{1}{2} - \frac{1}{3}x)x^{-\frac{1}{2}}e^{-\frac{1}{3}x} = 0$$

Dado que $\lim_{x\to\infty} h(x) = 0$ y $\lim_{x\to 0^+} h(x) = 0$, el valor máximo estará dado en algún punto crítico. En este caso, el valor de x tal que

$$\frac{1}{2} - \frac{1}{3}x = 0, \qquad x = \frac{3}{2}.$$

Entonces el valor máximo alcanzado por la función h es:

$$h(3/2) = 3\left(\frac{3}{2\pi e}\right)^{1/2}.$$

Este valor puede ser utilizado como la cota c del algoritmo. Luego, como Ahora, como

$$\frac{f(x)}{c\,g(x)} = \sqrt{\frac{2e}{3}}\,x^{1/2}e^{-x/3} \sim 1.34717$$

Así el algoritmo resulta:

```
def Gamma():
    while 1:
        U = 1-random()
        Y = - log(U) * 1.5
        V = random()
        if V < 1.3471 * Y ** 0.5 * exp(-Y / 3):
            return Y
```

Podríamos preguntarnos en el Ejemplo 6.5 si es razonable rechazar con una exponencial de igual media que la Gamma a generar. Para ello, deberíamos encontrar una cota de:

$$\frac{f(x)}{\lambda e^{-\lambda x}}$$

y determinar el valor de λ para la cual la cota es mínima. Notemos que:

$$\frac{f(x)}{\lambda e^{-\lambda x}} = \frac{Kx^{1/2}e^{-(1-\lambda)x}}{\lambda},$$

y esta función es no acotada si $\lambda \geq 1$. Luego corresponde analizar los casos en que $0 < \lambda < 1$.

Analizando puntos críticos, podemos ver que el punto de máximo está en:

$$x = \frac{1}{2(1-\lambda)}, \qquad 0 < \lambda < 1.$$

y el valor máximo es igual a

$$c_\lambda = \frac{K}{\lambda}(2(1-\lambda))^{-1/2}e^{-1/2}.$$

El valor máximo c_λ será mínimo si $\lambda(1-\lambda)^{1/2}$ es máximo, y esto ocurre si $\lambda = \frac{2}{3}$. Por lo tanto $\lambda = \frac{2}{3}$ minimiza el valor de la cota c.

6.4. Simulación de variables aleatorias normales

Las variables aleatorias normales tienen un uso extensivo, por lo cual es importante obtener un buen método para simularlas. En primer lugar, recordemos que si $Z \sim N(0,1)$, entonces

$$X = \sigma \cdot Z + \mu \sim N(\mu, \sigma).$$

Por lo tanto es suficiente tener un buen método para generar variables normales estándar.

El **método de la transformada inversa** no es una opción, puesto que la función de distribución acumulada:

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}}\, dt$$

no tiene una expresión cerrada y por lo tanto no se conoce una forma explícita de su inversa.

6.4.1. Por composición usando $|Z|$

Una observación es que la función de densidad de Z es par, por lo cual puede intentarse un método para generar $|Z|$ y usar luego un método de composición. Más precisamente, consideramos $U \sim \mathcal{U}(0,1)$ y X la variable aleatoria dada por

$$X = \begin{cases} |Z| & \text{si } U < 0.5 \\ -|Z| & \text{si } U \geq 0.5, \end{cases}.$$

Entonces se cumple que:

$$\begin{aligned} P(X \leq a) &= P(U < 0.5,\ |Z| \leq a) + P(U > 0.5,\ -|Z| \leq a) \\ &= \frac{1}{2}P(|Z| \leq a) + \frac{1}{2}P(-|Z| \leq a). \end{aligned}$$

Con un análisis de casos $a \geq 0$ y $a < 0$ podemos ver que $P(X \leq a) = P(Z \leq a)$, es decir que X tiene distribución normal.

Así Z es la mezcla o composición de las variables $|Z|$ y $-|Z|$, ambas con ponderación $1/2$:

$$F_Z(x) = 0.5 \cdot F_{|Z|}(x) + 0.5 \cdot F_{-|Z|}(x),$$

Luego, conociendo un algoritmo para la generación de $|Z|$ tenemos el siguiente método para simular Z:

```
def Normal_composicion():
    Generar |Z|
    if random() < 0.5:
        return |Z|
    else:
        return -|Z|
```

Es necesario entonces un método para generar $|Z|$. Su densidad está dada por:

$$f_{|Z|}(x) = \begin{cases} \frac{2}{\sqrt{2\pi}} e^{-x^2/2} & x > 0 \\ 0 & c.c. \end{cases}$$

Una posibilidad es utilizar el **método de rechazo** con una exponencial, por ejemplo, $X = \mathcal{E}(1)$. Denotamos g a la densidad de X. Así, debemos encontrar una cota para:

$$\frac{f_{|Z|}(x)}{g(x)} = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}+x}.$$

Esta función alcanza un máximo en el punto en que el exponente es máximo, lo cual ocurre en $x = 1$. El valor máximo y por consiguiente la cota mínima c está dada por:

$$c = \sqrt{\frac{2e}{\pi}} \approx 1.32$$

Para describir el algoritmo, calculamos:

$$\frac{f(x)}{c\, g(x)} = \exp\left\{-\frac{(x-1)^2}{2}\right\}.$$

Luego el método de rechazo simula $Y_1 \sim \mathcal{E}(1)$, U uniforme, e itera hasta que:

$$U \leq \exp\left\{-\frac{(Y_1-1)^2}{2}\right\},$$

es decir:

$$\log(U) \leq -\frac{(Y_1-1)^2}{2} \qquad \text{o equivalentemente} \qquad -\log(U) \geq \frac{(Y_1-1)^2}{2}.$$

Dado que $Y_2 = -\log(U)$ genera una exponencial $\mathcal{E}(1)$, el algoritmo por el método de rechazo para generar Z se traduce en:

```
def Normal_rechazo(mu, sigma):
    while True:
        Y1 = -log(random())
        Y2 = -log(random())
        if Y2 >=(Y1-1) ** 2 / 2:
            if random() < 0.5:
                return Y1 * sigma + mu
            return -Y1 * sigma + mu
```

Recordemos que una variable con distribución exponencial posee la propiedad de **falta de memoria**:

$$P(X > s + t \mid X > t) = P(X > s).$$

Es decir, condicional a que la variable X sea mayor a t, la cantidad que excede a este valor también tiene distribución exponencial. En particular, en el paso del algoritmo anterior en que se acepta el valor de Y_1, ocurre que Y_2 supera a $(Y_1-1)^2/2$ y por lo tanto la cantidad que excede:

$$Y_2 - (Y_1 - 1)^2/2$$

tiene distribución exponencial $\mathcal{E}(1)$.

De esta manera, el método de rechazo para generar $Z \sim N(0,1)$ permite generar también una exponencial. Esta exponencial podría ser utilizada en pasos sucesivos como el primer valor de Y_1, y así disminuir en 1 el número de exponenciales generadas. Como el método tiene un promedio de $c \sim 1.32$ iteraciones, el número de exponenciales necesarias será del orden de

$$2 \cdot 1.32 - 1 = 1.64.$$

6.4.2. Método polar

Otro método eficiente para generar variables normales es el **método polar**. El nombre se refiere al uso de coordenadas polares para referenciar puntos del plano:

$$x = r\cos\theta, \qquad y = r\,\mathrm{sen}\,\theta, \quad r \geq 0, \quad ,0 \leq \theta < 2\pi. \tag{6.1}$$

Consideramos dos variables aleatorias X e Y, normales estándar, independientes. Dado que toman cualquier valor real, entonces el par (X, Y) denota a algún punto del plano. Las coordenadas polares correspondientes a este punto, R y Θ, verifican:

$$R^2 = X^2 + Y^2, \qquad \Theta = \arctan\frac{Y}{X}.$$

El objetivo es analizar la distribución de las variables R^2 y Θ y simular estas variables para obtener dos normales estándar.

La función de densidad conjunta de X e Y está dada por:

$$f_{X,Y}(x,y) = f_X(x) \cdot f_Y(y) = \frac{1}{2\pi}\, e^{-(x^2+y^2)/2}.$$

La transformación de coordenadas (d,θ) a (x,y) está dada por:

$$d = x^2 + y^2, \qquad \theta = \arctan\frac{y}{x},$$

y por lo tanto la matriz jacobiana de esta transformación es:

$$\begin{pmatrix} \frac{\partial}{\partial x}(x^2+y^2) & \frac{\partial}{\partial y}(x^2+y^2) \\ \frac{\partial}{\partial x}(\arctan\frac{y}{x}) & \frac{\partial}{\partial y}(\arctan\frac{y}{x}) \end{pmatrix} = \begin{pmatrix} 2x & 2y \\ -\frac{y}{x^2+y^2} & \frac{x}{x^2+y^2} \end{pmatrix}$$

con determinante jacobiano igual a 2. Por lo tanto, la densidad conjunta de R^2 y Θ satisface:

$$f_{R^2,\Theta}(d,\theta) = \frac{1}{2}\, f_{X,Y}(x,y) = \frac{1}{4\pi}\, e^{-d/2}.$$

Así, observando que θ toma valores en $[0,2\pi)$ y $d > 0$, formalmente resulta:

$$f_{R^2,\Theta}(d,\theta) = \underbrace{\frac{1}{2\pi}\mathbb{I}_{[0,2\pi)}(\theta)}_{\Theta\sim U(0,2\pi)} \cdot \underbrace{\frac{1}{2}\, e^{-d/2}\, \mathbb{I}_{[0,\infty)}(d)}_{R^2\sim\mathcal{E}(\frac{1}{2})}.$$

Es decir, R^2 y Θ resultan ser variables aleatorias independientes, con distribución exponencial $\mathcal{E}(\frac{1}{2})$ y uniforme $\mathcal{U}(0,2\pi)$, respectivamente. El **Método polar** genera dos variables normales estándar, y el algoritmo es como sigue. Corresponde a la función normal.gauss() de Python.

```
def MetodoPolar():
    Rcuadrado = -2 * log( 1 - random() )
    Theta= 2 * Pi * random()
    X= sqrt(Rcuadrado) * cos(Theta)
    Y= sqrt(Rcuadrado) * sen(Theta)
    return (X * sigma + mu, Y * sigma + mu)
```

6.4.3. Transformaciones de Box-Muller

Las transformaciones:

$$\begin{cases} X = \sqrt{-2\log(U_1)}\cos(2\pi U_2) \\ Y = \sqrt{-2\log(U_1)}\,\mathrm{sen}(2\pi U_2) \end{cases} \tag{6.2}$$

se denominan **transformaciones de Box-Muller**. Una desventaja de estas transformaciones es que requieren el cálculo de dos funciones trigonométricas: seno y coseno. Para mejorar este paso, notemos que si generamos uniformemente puntos en el círculo unitario, y (V_1, V_2) son las coordenadas de un punto aleatorio en este círculo, entonces para cada r, $0 < r < 1$ se cumple que:

$$P(V_1^2 + V_2^2 \leq r) = P(\sqrt{V_1^2 + V_1^2} \leq \sqrt{r}) = \frac{\pi\, r}{\pi} = r,$$

y para α, $0 \leq \alpha < 2\pi$,

$$P(0 < \arctan \frac{V_2}{V_1} < \alpha) = \frac{\alpha/2}{\pi} = \frac{1}{2\pi}\,\alpha.$$

Así, las variables aleatorias $S^2 = V_1^2 + V_2^2$ y $\Theta = \arctan \frac{V_2}{V_1}$ resultan uniformemente distribuidas en $(0, 1)$ y $(0, 2\pi)$ respectivamente. Además por propiedades geométricas no es difícil ver que son independientes:

$$P(S^2 \leq r,\ \Theta < \alpha) = r \cdot \frac{\alpha}{2\pi}.$$

Así tenemos que:

$$\cos\Theta = \frac{V_1}{\sqrt{V_1^2 + V_2^2}} = \frac{V_1}{\sqrt{S^2}}, \qquad \operatorname{sen}\Theta = \frac{V_2}{\sqrt{V_1^2 + V_2^2}} = \frac{V_2}{\sqrt{S^2}}. \tag{6.3}$$

En resumen, para generar $\cos\Theta$ y $\operatorname{sen}\Theta$ debemos:

1. Generar un punto aleatorio (V_1, V_2) del círculo unitario.
2. Calcular $\cos\Theta$ y $\operatorname{sen}\Theta$ usando (6.3).

La generación de puntos aleatorios en el círculo se reduce a generar pares de puntos (V_1, V_2) en el cuadrado $[-1, 1] \times [-1, 1]$ hasta obtener uno en el círculo unitario. Las **transformaciones de Box-Muller** se reescriben entonces como:

$$X = \sqrt{-2\log U}\,\frac{V_1}{\sqrt{(V_1^2 + V_2^2)}} \tag{6.4}$$

$$Y = \sqrt{-2\log U}\,\frac{V_2}{\sqrt{(V_1^2 + V_2^2)}} \tag{6.5}$$

Ahora, como $S^2 = V_1^2 + V_2^2$ es uniforme en $(0, 1)$ y es independiente de Θ, puede ser utilizado como U en (6.4) y (6.5), resultando las ecuaciones:

$$X = \sqrt{-2\log U} \cdot \frac{V_1}{\sqrt{U}} = V_1 \cdot \sqrt{\frac{-2\log U}{U}} \tag{6.6}$$

$$Y = \sqrt{-2\log U} \cdot \frac{V_2}{\sqrt{U}} = V_2 \cdot \sqrt{\frac{-2\log U}{U}} \tag{6.7}$$

Finalmente, el método polar con las transformaciones (6.6) para simular dos variables normales resulta:

```
def Polar_Box_Muller(mu,sigma):
    #Generar un punto aleatorio en el círculo unitario.
    while True:
        V1, V2 = 2 * random()-1, 2 * random()-1
        if V1 ** 2 + V2 ** 2 <= 1:
            S = V1 ** 2 + V2 ** 2
            X = V1 * sqrt(-2 * log(S) / S)
            Y = V2 * sqrt(-2 * log(S) / S)
            return ( X * sigma + mu, Y * sigma + mu)
```

6.4.4. Método de razón entre uniformes

El método de razón entre uniformes se aplica en la generación de variables aleatorias continuas. Para ello, dada X una variable aleatoria continua con función de densidad f se considera el conjunto del plano C_f dado por:

$$C_f = \left\{(u,v) \mid 0 < u < \sqrt{f(v/u)}\right\}.$$

Kinderman y Monahan[1] demostraron el siguiente resultado:

Teorema 6.2. Si U y V son variables aleatorias continuas tales que (U,V) está uniformemente distribuida en C_f entonces la variable aleatoria $X = \frac{V}{U}$ tiene distribución f.

Para ver esto, consideremos U y V como en el teorema. Su distribución conjunta está dada por

$$f_{U,V}(u,v) = \frac{1}{|C_f|} \cdot \mathbb{I}_{C_f}(u,v),$$

donde $\mathbb{I}_{C_f}$ denota la función indicadora del conjunto C_f. Sea $T : C_f \mapsto \mathbb{R}^2$ dada por:

$$T(u,v) = \left(\frac{v}{u}, u\right) = (x,y).$$

T es biyectiva sobre su imagen y el valor absoluto del jacobiano de esta transformación es:

$$\left|\det\begin{pmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{pmatrix}\right| = \left|\det\begin{pmatrix} -\frac{v}{u^2} & \frac{1}{u} \\ 1 & 0 \end{pmatrix}\right| = |-\frac{1}{u}| = \frac{1}{y}.$$

Luego, si $T(U,V) = (X,Y)$, la densidad conjunta de X e Y está dada por

$$f_{X,Y}(x,y) = y \cdot f_{U,V}(T^{-1}(x,y)) = \begin{cases} \frac{1}{|C_f|}\, y & \text{si } 0 < y < \sqrt{f(x)} \\ 0 & \text{en otro caso,} \end{cases}$$

[1] A. J. KINDERMAN AND J. F. MONAHAN, *Computer generation of random variables using de ratio of uniform deviates*. ACM Transactions on Mathematical Software, Vol 3, No. 3, September 1977, Pages 257-260

y así la densidad marginal de $X = V/U$ es:

$$f_X(x) = \int_0^{\sqrt{f(x)}} \frac{1}{|C_f|}\, y\, dy = \frac{1}{2|C_f|} f(x).$$

En particular, para que f_X sea densidad debe ser $|C_f| = \frac{1}{2}$, y queda demostrado que $f_X(x) = f(x)$.

De esta manera, para generar una variable aleatoria X con densidad f puede seguirse el siguiente algoritmo:

1. Generar un vector aleatorio (U, V) uniformemente en un rectángulo $(0, c) \times (a, b)$ que contenga a C_f.

2. Si $U^2 < f(V/U)$ devolver $Z = \frac{V}{U}$, sino volver al paso 1.

Para el caso de Z con distribución normal estándar, el conjunto C_f está dado por

$$C_f = \{(u, v) \mid 0 < u < \frac{1}{\sqrt[4]{2\pi}}\, e^{-\frac{v^2}{4u^2}}\}.$$

Para simplificar la escritura, llamamos $C = \sqrt[4]{2\pi}$. Así, un par (u, v) pertenece a C_f si y sólo si

$$\ln(u \cdot C) < -\frac{v^2}{4u^2} \qquad \text{si y sólo si} \qquad 0 \le v^2 < -4u^2 \ln(u \cdot C), \tag{6.8}$$

La condición (6.8) requiere que $0 < u \le 1/C$ porque de lo contrario $\ln(u \cdot C) > 0$ o el logaritmo no está definido. Notemos que la función:

$$G(u) = -4u^2 \ln(u \cdot C), \qquad 0 < u \le 1/C$$

alcanza un máximo en el valor u tal que $G'(u) = 0$. Dado que:

$$G'(u) = -4(2u \ln(u \cdot C) + u) = -4u \cdot (2 \ln(u \cdot C) + 1),$$

el máximo se alcanza en $u = e^{-0.5}/C$ y el valor máximo es

$$G(e^{-0.5}/C) = -4\frac{1}{eC^2}(-0.5) = \frac{2}{eC^2}.$$

Así la ecuación (6.8) implica que $|v| < \frac{2}{C\sqrt{2e}}$. Sea $b = \frac{2}{C\sqrt{2e}}$. Entonces todo par (u, v) en el conjunto C_f satisface que $0 < C \cdot u < 1$ y $-b < v < b$, o lo que es lo mismo C_f está comprendido dentro del rectángulo:

$$R = [0, \frac{1}{C}] \times [-b, b].$$

Así, para generar pares (U, V) uniformemente en C_f podemos generar una secuencia de pares (U, V) uniformes en R hasta que cumplan la condición (6.8). Equivalentemente, generamos pares (U_1, U_2) uniformes en $[0, 1) \times [0, 1)$ y tomamos $U = U_1/C$, $V = 2b \cdot (U_2 - 0.5)$. Luego chequeamos (6.8) observando que

$$Z = \frac{V}{U} = \frac{\frac{4}{C\cdot(2e)^{1/2}}(U_2 - 0.5)}{\frac{U_1}{C}} = \frac{4}{(2e)^{1/2}}\frac{U_2 - 0.5}{U_1},$$

y también que

$$U^2 < f(Z) \quad \text{es equivalente a } U_1^2 < e^{-Z^2/2} \quad \text{o } Z^2 < -4\ln(U_1).$$

Los pasos son entonces:

1. Generar $U_1, U_2 \sim \mathcal{U}(0, 1)$.
2. Calcular $Z = (2e)^{-1/2}\frac{U_2-0.5}{U_1}$.
3. Si $Z^2 < -4\ln(U_1)$ devolver Z. Si no volver al paso 1.

El siguiente código es la implementación en Python del método de razón de uniformes para la función random.normalvariate(mu, sigma).

```
from math import exp
NV_MAGICCONST = 4 * exp(-0.5) / sqrt(2.0)
def normalvariate(mu, sigma):
    while 1:
        u1 = random()
        u2 = 1.0 - random()
        z = NV_MAGICCONST * (u1 - 0.5) / u2
        zz = z * z / 4.0
        if zz <= -log(u2):
            break
    return mu + z * sigma
```

La probabilidad de aceptación en este algoritmo es el cociente entre el área de C_f y el área de R:

$$\frac{\text{Área de } C_f}{\text{Área de } R} = \frac{0.5}{\frac{1}{\sqrt{2\pi}}\frac{4}{\sqrt{2e}}} = \frac{\sqrt{\pi e}}{4} = 0.7306$$

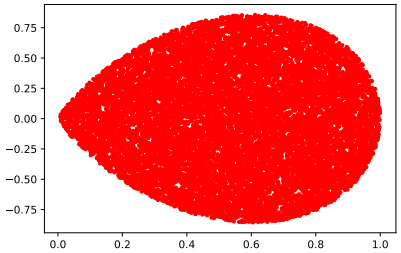

Figura 6.1: Región C_f para la distribución normal estándar

6.5. Generación de un Proceso de Poisson

6.5.1. Procesos de Poisson homogéneos

En un proceso de Poisson homogéneo de razón λ, los tiempos de llegada entre eventos sucesivos son exponenciales con parámetro λ. Así, para generar los primeros n eventos de un Proceso de Poisson homogéneo, generamos exponenciales $X_i \sim \mathcal{E}(\lambda)$, $1 \leq i \leq n$:

- Primer evento: al tiempo X_1.
- j-ésimo evento: al tiempo $X_1 + X_2 + \cdots + X_j$, para $1 \leq j \leq n$.

Para generar los eventos en las primeras T unidades de tiempo, generamos eventos hasta que $j+1$ excede a T. El siguiente algoritmo devuelve NT al número de eventos que ocurren hasta el tiempo T. El i-ésimo elemento de $Eventos$ ($Eventos[i-1]$) indica el tiempo en que ocurre el evento i, $1 \leq i \leq NT$:

```
def eventosPoisson(lamda,T):
    t = 0
    NT = 0
    Eventos = []
    while t < T:
        U = 1 - random()
        t += - log(U) / lamda
        if t <= T:
            NT += 1
            Eventos.append(t)
    return NT, Eventos
```

Recordemos que a partir de una variable $Gamma(n, \lambda^{-1})$ y $n-1$ uniformes, es posible generar n exponenciales. Para ello, ordenamos las $n-1$ uniformes en forma creciente y los valores de estas uniformes indican los tiempos de arribo.

Por otro lado, se puede probar el siguiente resultado:

Proposición 6.2. Dado $N(T)$, la distribución de los tiempos de arribo en un Proceso de Poisson homogéneo de intensidad λ es uniforme en $(0, T)$.

Demostración. En el material complementario.

□

Por lo tanto, un método alternativo para generar los tiempos de arribo hasta el tiempo T consiste en:

- Generar una variable aleatoria Poisson de media λT, y tomar $n = N(T)$.
- Generar n variables aleatorias uniformes $U_1, U_2, \ldots, U_n$.
- Ordenarlas: $U_{i_1} < U_{i_2} < \ldots < U_{i_n}$.
- Los tiempos de arribo son: $TU_{i_1}, TU_{i_2}, \ldots, TU_{i_n}$.

Si bien este algoritmo posee la ventaja de no generar una secuencia de exponenciales, requiere realizar un ordenamiento de $n = N(T)$ números.

6.5.2. Procesos de Poisson no homogéneos

Consideremos un proceso de Poisson no homogéneo con función de intensidad $\lambda(t)$, y sea λ tal que

$$\lambda(t) \leq \lambda.$$

En la Proposición 2.1hemos visto que si $M(t)$ es un proceso homogéneo con intensidad λ, y consideramos $N(t)$ el proceso que cuenta los eventos de $M(t)$ con probabilidad $\frac{\lambda(t)}{\lambda}$, entonces $N(t)$ resulta un proceso de Poisson no homogéneo con intensidad $\lambda(t)$. Esta propiedad nos permite proponer el siguiente algoritmo de generación de un proceso de Poisson no homogéneo con función de intensidad $\lambda(t)$:

```
def Poisson_no_homogeneo_adelgazamiento(T):
    'Devuelve el número de eventos NT y los tiempos en Eventos'
    'lamda_t(t): intensidad, lamda_t(t)<=lamda'
    NT = 0
    Eventos = []
    U = 1 - random()
    t = -log(U) / lamda
    while t <= T:
        V = random()
```

```
        if V < lamda_t(t) / lamda:
            NT += 1
            Eventos.append(t)
        t += -log(1 - random()) / lamda
    return NT, Eventos
```

Nuevamente, NT finaliza con el número de eventos hasta el tiempo T, es decir es $N(T)$, y $Eventos[0]$, $Eventos[1]$, ..., $Eventos[NT-1]$ son los tiempos de cada evento.

Notemos que el algoritmo es más eficiente cuánto más cerca esté λ de $\lambda(t)$, ya que en ese caso $\lambda(t)/\lambda$ es próximo a 1. Una mejora de este algoritmo es particionar el intervalo $(0,T)$ en subintervalos, y aplicar el algoritmo anterior en cada uno de ellos con un valor λ_i adecuado. Esto es, considerar k intervalos consecutivos $[t_{i-1}, t_i)$, $1 \leq i \leq k$ tales que:

$$0 = t_0 < t_1 < \ldots < t_k = T,$$

y valores $\lambda_1, \lambda_2, \ldots, \lambda_k$ que cumplan

$$\lambda(s) \leq \lambda_i, \qquad s \in [t_{i-1}, t_i).$$

Ejemplo 6.6. Consideremos un proceso de Poisson no homogéneo con función de intensidad

$$\lambda(t) = 2t + 1, \qquad 0 \leq t \leq 6.$$

Una posibilidad es implementar el algoritmo recién visto a través de la generación de un proceso de Poisson homogéneo con intensidad constante $\lambda = 13$. Pero también podemos subdividir el intervalo $[0, 6]$ en tres subintervalos:

$$I_1 = [0, 2], \qquad I_2 = (2, 4], \qquad I_3 = (4, 6],$$

y acotar $\lambda(t)$ con $\lambda_1 = 5$, $\lambda_2 = 9$ y $\lambda_3 = 13$ respectivamente.

Notemos que por la propiedad de falta de memoria de la exponencial, si $t \in I_i$ y $t - \frac{\ln(1-U)}{\lambda_i} \in I_j$ con $j > i$, entonces $\frac{\lambda_i}{\lambda_j} \cdot (t - t_i)$ es exponencial con parámetro λ_j. Esta propiedad resulta útil para aprovechar la última exponencial generada en cada subintervalo.

Así, para este ejemplo en particular, el algoritmo resulta:

```
def Poisson_adelgazamiento_mejorado(T):
    interv = [2,4,6] #T<=6
    lamda = [5, 9, 13]
    j = 0 #recorre subintervalos.
    t = -log ( 1 - random() ) / lamda[j]
    NT = 0
```

```
    Eventos = []
    while t <= T:
        if t <= interv[j]:
            V = random()
            if V < (2 * t + 1) / lamda[j]:
                NT += 1
                Eventos.append(t)
            t += -log(1 - random()) / lamda[j]
        else: #t > interv[j]
            t = interv[j] + (t - interv[j]) * lamda[j] / lamda[j + 1]
            j += 1
    return NT, Eventos
```

6.6. Método de aceptación y rechazo transformado

El método de aceptación y rechazo para simular una variable aleatoria X con densidad f se basa en simular una variable aleatoria Y con densidad g, y aceptar el valor Y generado con probabilidad $\frac{f(Y)}{cg(Y)}$ para cierto $c > 1$. El **método de rechazo transformado** se basa en la misma idea pero utiliza la inversa de la distribución acumulada de Y en lugar de la densidad.

Si g es la función de densidad de Y, entonces su función de distribución acumulada está dada por:

$$G(x) = \int_{-\infty}^{x} g(t)\,dt, \qquad x \in \mathbb{R}.$$

G toma valores en el intervalo $[0, 1]$ y al igual que para el método de la transformada inversa asumiremos además que su inversa está definida sobre $(0, 1)$ y la denotaremos H:

$$H(u) = G^{-1}(u), \qquad u \in (0, 1).$$

Hemos visto además que si U es una variable aleatoria con distribución uniforme en $(0, 1)$ entonces $H(U)$ tiene igual distribución que Y. Por lo tanto el método de aceptación y rechazo podría describirse con los siguientes pasos:

```
def Aceptacion_Rechazo_X():
    while 1:
        U = random()
        V = random()
        if V < f(H(U)) / (c * g(H(U)):
            return H(U)
```

Ahora bien, g es la densidad de Y, y por lo tanto $g(x) = G'(x)$. Además, por ser H la inversa de G tenemos que $G(H(u)) = (G \circ H)(u) = u$ y por lo tanto

$$(G \circ H)'(u) = 1 \tag{6.9}$$

Pero

$$(G \circ H)'(u) = G'(H(u)) \cdot H'(u) = g(H(u)) \cdot H'(u). \tag{6.10}$$

De (6.9) y (6.10) se sigue que

$$g(H(u)) = \frac{1}{H'(u)}.$$

Por lo tanto el método de aceptación y rechazo utilizando la transformada inversa resulta:

```
def Aceptacion_Rechazo_Transformado():
    while 1:
        U = random()
        V = random()
        if V < f(H(U)) * Hprima(U) / c: ## Hprima = derivada de H
            return H(U)
```

Ejemplo 6.7. Para ilustrar la aplicación del método consideremos nuevamente la simulación de la variable $Gamma(\frac{3}{2}, 1)$ como en el Ejemplo 6.5. Así, si se simula la variable X rechazando con una variable aleatoria exponencial $Y \sim \mathcal{E}(2/3)$ tenemos:

$$f(x) = \frac{2}{\sqrt{\pi}} x^{1/2}\, e^{-x}, \qquad g(x) = \frac{2}{3}\, e^{-\frac{2}{3}x}, \qquad x > 0$$

y la constante c dada por:

$$c = 3\sqrt{\frac{3}{2\pi e}}.$$

La función de distribución acumulada de Y es $G(x) = 1 - e^{-\frac{2}{3}x}$ para $x > 0$. Su inversa $H : (0,1) \mapsto \mathbb{R}^{>0}$ y su derivada están dadas por:

$$H(u) = -\frac{3}{2}\ln(1-u), \qquad H'(u) = \frac{3}{2(1-u)}.$$

Para el método de aceptación y rechazo transformado es necesario calcular $f(H(u)) \cdot H'(u)$, para $0 < u < 1$:

$$\begin{aligned} f(H(u)) \cdot H'(u) &= \frac{2}{\sqrt{\pi}} \left(-\frac{3}{2}\ln(1-u)\right)^{1/2} e^{\frac{3}{2}\ln(1-u)} \frac{3}{2(1-u)} \\ &= 3\sqrt{\frac{3}{2\pi}}\, (-\ln(1-u))^{1/2}\, (1-u)^{3/2} \frac{1}{1-u} \\ &= 3\sqrt{\frac{3}{2\pi}}\, (-\ln(1-u)\,(1-u))^{1/2}\,. \end{aligned}$$

Dividiendo por la constante c resulta:

$$\frac{f(H(u)) \cdot H'(u)}{3\sqrt{\frac{3}{2\pi e}}} = \frac{1}{3\sqrt{\frac{3}{2\pi e}}} \cdot 3\sqrt{\frac{3}{2\pi}} \left(-\ln(1-u)\,(1-u)\right)^{1/2} = e^{1/2} \left(-\ln(1-u)\,(1-u)\right)^{1/2}.$$

Así la implementación del método de aceptación y rechazo transformado resulta:

```
exp_un_medio = 1.6487212 ## exp(0.5)
def Gamma():
    while True:
        U = 1 - random()
        V = random()
        if V < exp_un_medio * sqrt( - log(U) * U)
            return - 1.5 * log(U)
```

6.6.1. Método de rechazo transformado con compresión

Una particularidad del método de rechazo transformado es que la función

$$\frac{1}{c} \cdot f(H(u)) \cdot H'(u), \qquad 0 < u < 1 \tag{6.11}$$

tiene su gráfico en el cuadrado unitario $[0,1] \times [0,1]$. El algoritmo consiste en generar puntos aleatorios (U, V) en este cuadrado; si el punto queda por debajo del gráfico de la función se devuelve el valor $H(U)$ y de lo contrario se repite el proceso. Ahora bien, si la variable Y utilizada para rechazar tiene una densidad próxima a la densidad de X, entonces por debajo del gráfico de la función en (6.11) se podrá ubicar un rectángulo con un área más o menos grande. Por ejemplo, para el caso del Ejemplo 6.7, el gráfico de la función dada en (6.11) y un posible rectángulo bajo el gráfico es como en la Figura 6.6.1.

Notemos que si en el algoritmo un punto (U, V) cae en el rectángulo entonces el valor $H(U)$ será necesariamente aceptado, y por lo tanto no es necesario computar $f(H(U))\, H'(U)$. De esta manera, si el rectángulo es de la forma $[u_1, u_2] \times [0, v_1]$ el algoritmo general puede modificarse como:

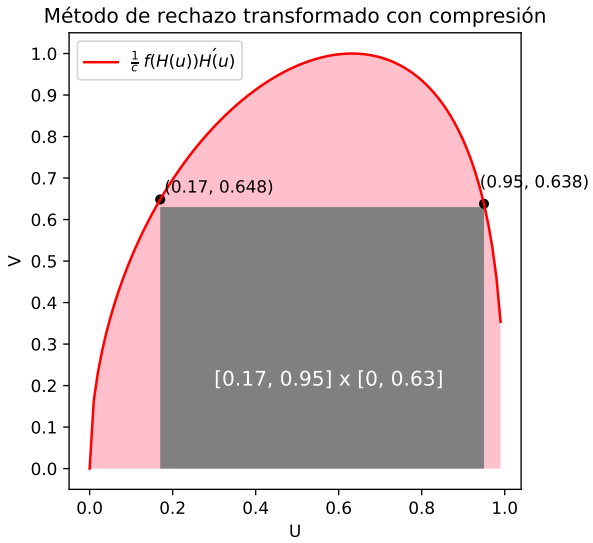

Figura 6.2: $\frac{1}{c}\, f(H(u)) \cdot H'(u) = e^{1/2}\left(-(1-u)\ln(1-u)\right)^{1/2}$

```
def Rechazo_con_compresion():
    while True:
        U = random()
        V = random()
        if u_1 <= U <= u_2 and V <= v_1:
            return H(U)
        if V < f(H(U)) * Hprima(U) / c:
            return H(U)
```

Así, si bien el número esperado de iteraciones del algoritmo para devolver un valor sigue siendo c, se logra reducir el número de evaluaciones $f(H(U))\, H'(U)$ en una cantidad proporcional al área del rectángulo, siendo ahora:

$$c \cdot (1 - (u_2 - u_1) \cdot v_1)\,.$$

En el Ejemplo 6.7, la constante c es aproximadamente $c \simeq 1.2573168$. Tomando $u_1 = 0.17$, $u_2 = 0.95$, $v_1 = 0.63$ como se muestra en la Figura 6.6.1, el número de evaluaciones promedio se reduce a:

$$1.2573167 \cdot (1 - 0.78 \cdot 0.63) \simeq 0.64.$$

En otras palabras, si para generar 100 números el algoritmo requiere realizar unas 126 iteraciones, aproximadamente caen en el rectángulo unos 62 ($\sim 126 \cdot 0.78 \cdot 0.63$) puntos y por lo tanto sólo en 64 de las 126 iteraciones será necesario evaluar $f(H(U)) \cdot H'(U)$.

En el artículo de Wolfgan Hörmann[2] se describe la aplicación de este método para el caso particular de la generación de una variable normal y de una variable Poisson, tomando como H un representante de una familia especial de funciones de distribución de variables continuas. Dado que una Poisson se trata de una variable discreta, se utiliza como densidad f el histograma normalizado, esto es, $f(x) = P(X = \lfloor x \rfloor)$ para $x \in \mathbb{R}$. El lenguaje Python implementa la generación de $X \sim \mathcal{P}(\lambda)$, para $\lambda \geq 10$ basándose en el trabajo de Hörman, mientras que para $\lambda < 10$ aplica suma de exponenciales como hemos descripto en la Sección 6.2.2.

[2]Wolfgang Hörman, *New Generators of Normal and Poisson Deviates Based on Transformed Rejection Method*. Department of Applied Statistics and Data Processing. Wirtschaftsuniversität Wien. Preprint Series. Preprint 4. (1992)