Generación de variables aleatorias continuas

Metodo de la transformada inversa

Este metodo requiere de conocer la funcion de distribución acomulada de la variable X a simular. Si X es absolutamente continua y *f* es su función de densidad, entonces su función de distribución acumulada es:

$$\int_{-\inf}^{x} f(t)dt$$

Vamos trabajar sobre las variables X tal que su f.d.m sea monótona creciente sobre el conjunto $F^{-1}(0,1) = x|0 < F(X) < 1$. De esta forma nos aseguramos que F(x) sea invertible en el (0,1).

> **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.

Con esta proposicion podemos dar un algoritmo basico de como funciona el metodo de la transformada inversa:

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

Eso no siempre es posible ya que puede ser que F^{-1} involucre funciones costosas o que no se pueda calcular explicitamente. En estos casos se busca expresar F como:

- distribucion del minimo o del maximo de VA indep
- suma de variables indep
- distribuciones condicionales, etc.

Simulacion de variable Exponencial

Si X es una variable aleatoria con distribucion exponencial de parametro $\lambda = 1$ entonces su funcion de densidad es:

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

luego la inversa de F sobre (0,1) es:

$$F^{-1}(u) = -\log(1-u), u \in (0,1)$$

Entonces el algoritmo de simulacion para una exponencial con $\lambda = 1$ es:

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

Vemos que si Y es $\frac{1}{\lambda}X$ entonces Y distribuye exponencial con parametro λ y el codigo es:

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

Simulacion de variable Poisson

Sabemos que en un proceso de Poisson de intensidad λ entonces N(t), es una variable aleatoria Poisson con media $\lambda.t$. Y ademas sabemos que el tiempo entre llegadas de eventos son aleatorias exponenciales con media $\frac{1}{\lambda}$.

N(1) es una variable Poisson con paramentro λ, y sabemos que los tiempos entre arribos en [0,1] son exponenciales. Entonces si simulamos *i* variables aleatorias de modo que:

$$X_1 + X_2 + .. + X_n \leq 1 \text{ y } X_1 + X_2 + .. + X_n + X_{n+1} > 1$$

Entonces n es el número de arribos hasta t=1. Y tenemos

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

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(1-lamda)
    while producto >= cota:
        producto *= 1 - random
        x += 1
    return x
```

Simulacion de variable Gamma(n, λ^{-1})

Sabemos que la suma de n variables aleatorias exponenciales, independientes, con parametro λ es una variable aleatoria con distribucion Gamma(n, λ^{-1}). Esta propiedad nos permite dar un algoritmo que a partir de exponenciales generamos una Gamma.

$$X = -\frac{1}{\lambda}\log(U_1 \cdot U_2 \cdots U_n)$$

Y el algoritmo es:

```
def Gamma(n, lamda):
    # genera gamma con parametros n y 1/lamda
    u = 1
    for _ in range(n):
        u *= 1 - random()
    return -log(u)/lamda
```

utilizando este metodo de generacion de exponenciales para generar una gamma, entonces podemos diseñar un algoritmo para generar n variables independientes exponenciales de parametro λ

```
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
```

Para el caso general tenemos que calcular un unico logaritmo y n-1 uniformes adicioneles.

```
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
```

Metodo de aceptacion y rechazo

Supongamos que queremos generar X que tiene funcion de densidad *f*:

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

Y que tengo un método para generar otra Y con denssidad *g* tq:

$$\frac{f(y)}{g(y)} \leq c \quad \text{para todo } y \in \mathbb{R} : f(y) \neq 0$$

Entonces el algoritmo de aceptacion y rechazo es:

```
def AceptacionRechazo():
    while True:
        y = G()
        u = random()
        if u < f(y) / (c*g(y)):
            return y
```

Al igual que en las variables aleatorias discretas, este metodo tiene densidad *f* y el numero de iteraciones es una variable geometrica con media *c*.

Para poder generar X rechazando contra una variable Y debemos poder acotar $\frac{f(x)}{g(x)}$ con una constante c. Por lo que vamos a tener que considerar la funcion:

$$h(x) = \frac{f(x)}{g(x)}, \quad \text{para } x \in \mathbb{R} : f(x) \neq 0$$

Para esto vamos a:

- encontrar los puntos criticos de h(x) (donde h'(x) = 0 o h(x) no esta definida)
- ver cuales corresponden a máximos locales
- Luego tomamos ese maximo como ***c***

Simulacion de variables aleatorias normales

Por composicion usando |Z|

Buscamos generar una variable que sea el valor absoluto de una distribucion normal estandar. A partir del metodo de aceptacion y rechazo contra una exponencial de media 1. Y tenemos:

$$\text{ABS de normal estandar} : f(x) = \frac{2}{\sqrt{2\pi}} e^{-x^2/2}$$
$$\text{Exponencial de media 1} : g(x) = e^{-x}$$
$$\text{entonces para encontrar la cota para el rechazo hacemos :}$$
$$\frac{f(x)}{g(x)} = \sqrt{(2\pi)} e^{x - x^2/2} \text{ y el maximo de esto esta cuando x =1}$$
$$c = \sqrt{2\pi e} \quad \text{y luego :}$$
$$\frac{f(x)}{cg(x)} = \exp - \frac{(x-1)^2}{2}$$

Entonces en nuestro algoritmo vamos a

- generar Y exponencial con media 1
- numero U aleatorio entre 0 y 1
- si U $\leq exp-(Y-1)^2/2$ enotnces X = Y.

notamos que en el 3er paso es lo mismo que hace $-\log(U) \leq (Y-1)^2/2$ y sabemos que $-\log(U)$ es una exponencial de media 1. Entonces el algoritmo se simplifica a

- generar Y1, Y2 exponenciales con media 1
- si $Y2 \geq (Y1-1)^2/2$ entonces X = Y1

```
def normal_estandar_ej():
    while True:
        y_1 = gen_exponencial(1)
        y_2 = gen_exponencial(1)
        if y_2 >= (y_1-1)**2/2:
            u = rd.random()
            if u <= 0.5:
                return y_1
            else:
                return -y_1
```

```
def normal_ej(mu, sigma):
    while True:
        y_1 = gen_exponencial(1)
        y_2 = gen_exponencial(1)
        if y_2 >= (y_1-1)**2/2:
            u = rd.random()
            if u <= 0.5:
                return y_1 * sigma + mu
            else:
                return -y_1 * sigma + mu
```

Metodo polar

En el metodo polar lo que se busca es generar dos variables normales a partir de las coordenadas polares de un punto (X,Y) en el plano, donde ambas son normales estandar e independientes. Entonces tendremos a $R^2 = X^2 + Y^2$ y $\Theta = \arctan(\frac{Y}{X})$. Utilizando el determinante jacobiano de la transformacion de coordenadas polares a cartesianas tenemos que:

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

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

```
def MetodoPolar():
    Rcuadrado = -2 * log(1 - random())
    theta = 2 * pi * random()
    X = sqrt(Rcuadrado) * cos(theta)
    Y = sqrt(Rcuadrado) * sin(theta)
    return X*sigma + mu, Y*sigma + mu
```

Transformaciones de Box-Muller

TODO

Método de razón entre uniformes

El metodo de razon entre uniformes busca generar una varaible X con funcion de densidad f tal que:

$$C_f = \{(u,v) | 0 < u < \sqrt{f(v/u)}\}$$

Si U y V son variables aleatorias continuas tales que (U,V) esta uniformemente distribuido en el conjunto C_f entonces la variable X = V/U tiene funcion de densidad f.

Luego los pasos del algoritmo son los siguientes:

1. Generar un vector aleatorio (U,V) uniformemente en el rectangulo (0, c) x (a, b) que contenga a C_f
2. Si $U^2 < f(V/U)$ entonces devolver X = V/U, sino volver a 1.

Para el caso de la normal estandar $C_f = \{(u,v) | 0 < u < \frac{1}{\sqrt[4]{2\pi}} e^{-v^2/4u^2}\}$

luego un par solo pertenece a C_f si ($C = \sqrt[4]{2\pi}$):

$$\ln(u \cdot C) < -\frac{v^2}{4u^2} \rightarrow 0 \leq v^2 < -4u^2 \ln(u \cdot C)$$

Esto requiere que $0 < u \leq \frac{1}{C}$.ademas $-4u^2 \ln(u \cdot C)$Toma maximo en $u = \frac{e^{-0.5}}{C}$ y el valor maximo es $\frac{2}{eC^2}$

Esto nos dice que $|v| < \frac{2}{C\sqrt{2e=b}}$, y vimos antes que $0 < C \cdot u < 1$. Entonces C_f esta comprendido en $R = [0, \frac{1}{C}] \times [-b, b]$

```
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
```

Generación de un proceso de Poisson

Recordamos que en un proceso de Poisson de parametro λ los tiempos de llegada entre eventos sucesivos son exponenciales con parametro λ. Entonces de esta forma si queremos generar los primeros n eventos tenemos que generar exponenciales $X_i \, \mathrm{E}(\lambda), 1 \leq i \leq n$

- Primer evento: $T_1 = X_1$
- j-esesimo evento: $X_1 + X_2 + \cdots + X_j$

Entonces para generar eventos hasta T, generamos evento hasta que lo generado en j+1 exceda a T. Recordamos que la forma de generar una exponencial con parametro λ es $-\log U/\lambda$

Entonces el codigo queda:

```
def eventos_poisson(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
```

NT es la cantidad de eventos hasta el tiempo T, y en Eventos se devuelve el tiempo cuando ocurrio el i-esimo evento.

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.

Generación de procesos de Poisson no homogéneos

Vimos que M(t) es un proceso homogéneo con intensidad λ y consideramos N(t) el proceso de poisson que cuenta los eventos de M(t) con probabilidad $\frac{\lambda(t)}{\lambda}$. enotnces N(t) es un proceso de Poisson no homogéneo con intensidad $\lambda(t)$.

```
def Poisson_no_homogeneo_adelgazamiento(T):
    'Devuelve el n´umero 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
```

Una alternativa para hacerlo mas eficiente, es particionar el intervalo [0,T] en subintervalos. Y acotar cada intervalo con un λ diferente.