TP 2.2 - GENERADORES PSEUDOALEATORIOS CON DISTINTAS DISTRIBUCIONES DE PROBABILIDAD

Antonelli, Nicolás
Departamento de Ingeniería en Sistemas
Universidad Tecnológica Nacional - FR Rosario
Rosario, Zeballos 1341
niconelli2@gmail.com

Acciarri, Joshua
Departamento de Ingeniería en Sistemas
Universidad Tecnológica Nacional - FR Rosario
Rosario, Zeballos 1341
acciarrijoshua@gmail.com

11 de junio de 2020

ABSTRACT

Los generadores de números pseudoaleatorios uniformes en informática son los más populares (e importantes) a la hora de trabajar con variables aleatorias, pero ésta no es la única distribución que estos números pueden tener. ¿Cómo se pueden generar números pseudoaleatorios en base a otras distribuciones de probabilidad? ¿Cómo es un algoritmos sencillo de implementar para generarlos manualmente? ¿Realmente se comportan estos números como deberían? En el presente trabajo responderemos estas preguntas.

***K*eywords** Números Pseudoaleatorios · Distribuciones · Generadores · Tests · Artículo Científico

1. Introducción

Un número pseudoaleatorio es un número generado en un proceso determinístico, pero que se comporta como si hubiese sido generado al azar, aunque realmente no sea así. Una buena secuencia de números pseudoaleatorios no presenta ningún patrón o regularidad aparente desde el análisis estadístico, y debe ser capaz de superar Tests de rendimiento según sea la distribución que se busca que tengan éstas secuencias.

Cabe aclarar que no importa cual sea la distribución de probabilidad que necesitamos que tengan los números pseudoaleatorios, es posible generarlos utilizando procedimientos basados puramente en estadística computacional básica que se traducen a un sencillo algoritmo para cada una de ellas.

2. Marco Teórico Principal

2.1. Verdaderos Números Aleatorios

Los números completamente aleatorios (no determinísticos) se basan en alguna fuente de aleatoriedad física que puede ser teóricamente impredecible (cuántica) o prácticamente impredecible (caótica), como por ejemplo *random.org* [1] que genera aleatoriedad a través de ruido atmosférico, o ERNIE que usa ruido térmico en transistores, el mismo se utiliza en la lotería de bonos de Reino Unido (apreciándose la confiabilidad del mismo).

Hablamos con un poco más de detalle sobre los verdaderos números aleatorios en otro documento (TP 2.1), si es de interés consultarlo [7].

2.2. Generadores Pseudoaleatorios

Un generador pseudoaleatorio de números (GPAN) es un algoritmo que produce una sucesión de números que es una muy buena aproximación a un conjunto aleatorio de números. La sucesión no es exactamente aleatoria en el sentido de que queda completamente determinada por un conjunto relativamente pequeño de valores iniciales, llamados el estado del GPAN. La mayoría de los algoritmos de generadores pseudoaleatorios producen sucesiones que poseen una distribución uniforme según varios tipos de pruebas[3]. Las clases más comunes de estos algoritmos son generadores lineales congruentes. Entre los desarrollos más recientes de algoritmos pseudoaleatorios se encuentran Blum Blum Shub, Fortuna, el Mersenne-Twister (que utiliza Python), entre otros.

Hablamos con mucho más detalle de los diferentes tipos de generadores pseudoaleatorios más conocidos y sus características en otro documento (TP 2.1), si es de interés conocer más acerca de éstos, consultarlo [7].

2.3. Conceptos Fundamentales sobre Distribuciones de Probabilidad para Simulaciones

2.3.1. Variables Aleatorias

Una *variable aleatoria* [5] es una función que asigna a cada elemento del espacio muestral un número real. Los valores posibles de una variable aleatoria pueden representar los posibles resultados de un experimento aún no realizado, o los posibles valores de una cantidad cuyo valor actualmente existente es incierto.

Una variable aleatoria es *continua* cuando el recorrido de la variable aleatoria es un intervalo real (matemáticamente no es factible asignar una probabilidad positiva a cada elemento del recorrido de la variable aleatoria).

Una variable aleatoria es *discreta* si su recorrido es un conjunto finito o infinito numerable (susceptible de ser contado).

Algunas variables aleatorias con distribución de probabilidad de tipo continua:

- Uniforme
- Normal
- Exponencial
- Gamma (Y variantes como Earlang, Chi^2, Beta ...)

Algunas variables aleatorias con distribución de probabilidad de tipo discreta:

- Binomial
- Pascal (Binomial Negativa)
- Poisson
- Hipergeométrica
- Empírica Discreta

Una variable aleatoria de cierta distribución, tiene asociado una *Función de distribución acumulada (FDA)*, y si la variable es continua también tendrá asociada una *Función de densidad de probabilidad (FDP)*.

2.3.2. Media y Varianza

La media aritmética es un promedio estándar que a menudo se denomina promedio. Se puede calcular de la siguiente manera:

$$E[X] = \mu_x = \sum_{i=1}^{k} x_i p_x(x_i) \tag{1}$$

La varianza es una medida de dispersión que representa la variabilidad de una serie de datos respecto a su media. Se puede calcular como:

$$V[X] = \sigma^2 = \sum_{i=1}^{k} (x_i - \mu_x)^2 p_x(x_i) \tag{2}$$

con $p_x(x_i)$ igual a la probabilidad de que la variable aleatoria X tome el valor x_i

2.3.3. Función de Distribución Acumulada

En teoría de la probabilidad y en estadística, la función de distribución acumulada (FDA) [9] asociada a una variable aleatoria real X sujeta a cierta ley de distribución de probabilidad, es una función matemática de la variable real x que describe la probabilidad de que X tenga un valor menor o igual que x. Es decir: $F(x) = P\{X \leq x\}$.

Para una variable aleatoria continua: $F(x) = \int_{-\infty}^{x} f(t)\, dt$, donde $f(t)$ es la función de densidad de probabilidad de la distribución (la cual explicamos abajo).

Para una variable aleatoria discreta: $F(x) = \sum_{x_i \leq x} f(x_i)$, donde $f(x_i)$ es la función de probabilidad de la distribución.

2.3.4. Función de Densidad de Probabilidad

En teoría de la probabilidad, la función de densidad de probabilidad (FDP) continua [10] describe la probabilidad relativa según la cual una variable aleatoria tomará determinado valor. La probabilidad de que la variable aleatoria caiga en una región específica del espacio de posibilidades estará dada por la integral de la densidad de esta variable entre uno y otro límite de dicha región. Esta función es positiva a lo largo de todo su dominio, y su integral sobre todo el espacio es de valor unitario.

Según la definición que dimos en función de densidad acumulada (FDA), se puede despejar la FDP a la siguiente expresión: $f(x) = \frac{d}{dx}F(x)$, y decimos que una variable aleatoria X tiene densidad f, siendo f una función no-negativa, continua e integrable, si: $P\{a \leq X \leq b\} = \int_a^b f(x)\, dx$.

2.3.5. El método de la transformación inversa

Si deseamos generar los valores x_i de las variables aleatorias a partir de cierta estadística de población cuya función de densidad este dada por $f(x)$ debemos en primer lugar obtener la función de distribución acumulativa $F(x)$. Puesto que $F(x)$ se define sobre el rango de 0 a 1, podemos generar números aleatorios distribuidos uniformemente y además hacer $F(x) = r$. Resulta claro cómo queda x determinada unívocamente por $r = F(x)$. Luego, para cualquier valor particular de r que generemos aleatoriamente, por ejemplo r_0 siempre es posible encontrar el valor de x (en este caso x_0) que corresponde a r_0 debido a la función inversa de F si ésta es conocida. Esto es:

$$x_0 = F^{-1}(r_0) \tag{3}$$

donde $F^{-1}(x)$ es la transformación inversa [6] de r sobre el intervalo unitario en el dominio de x. Si generamos números aleatorios uniformes correspondientes a una $F(x)$ dada, podemos resumir matemáticamente este método como sigue:

$$r = F(x) = \int_{-\infty}^{x} f(t)\, dt \tag{4}$$

entonces

$$P(X \leq x) = F(x) = P[r \leq F(x)] = P[F^{-1}(r) \leq x] \tag{5}$$

y consecuentemente $F^{-1}(r)$ es una variable que tiene a $f(x)$ como función de densidad de probabilidad. Este criterio equivale a resolver la ecuación (2) en x, en términos de r. Este procedimiento lo realizaremos para algunas de las siguientes distribuciones como se podrá observar.

2.3.6. Ensayos de Bernoulli

En la teoría de probabilidad y estadística, **un ensayo de Bernoulli** [20] es un experimento aleatorio en el que sólo se pueden obtener dos resultados (habitualmente etiquetados como éxito y fracaso). Se denomina así en honor a Jakob Bernoulli. Desde el punto de vista de la teoría de la probabilidad, estos ensayos están modelados por una variable aleatoria que puede tomar sólo dos valores (0 y 1). Habitualmente, se utiliza el 1 para representar el éxito.

Si p es la probabilidad de éxito, entonces el valor del valor esperado de la variable aleatoria es p y su varianza $p(1-p)$. Los *Procesos de Bernoulli* son los que resultan de la repetición en el tiempo de ensayos de Bernoulli independientes pero idénticos. Utilizaremos este concepto en distribuciones discretas.

3. Métodos para generar números pseudoaleatorios en diferentes distribuciones

En esta sección introduciremos la definición teórica de muchas distribuciones continuas y discretas junto a varias características de cada una, y partiremos de allí para llegar paso a paso al código en *Python* [15] mínimo necesario para generar un conjunto de números pseudoaleatorios que cumplan determinada distribución, es decir un algoritmo que sea sencillo de implementar. Luego, explicaremos los tests a los que fueron sometidos los números generados con dichos algoritmos obtenidos de aplicar los conceptos descritos en esa sección.

3.1. Distribuciones de Probabilidad Continuas

3.1.1. Distribución Uniforme

Decimos que una variable aleatoria X se distribuye uniformemente en el intervalo $[a,b]$ y notamos $X \sim \mathscr{U}(a,b)$ cuando su función de densidad de probabilidad es:

$$f_x(x) = \begin{cases} \frac{1}{b-a} & \text{si } x \in [a,b] \\ 0 & \text{en otro caso} \end{cases} \tag{6}$$

El hecho de que una variable aleatoria tenga un comportamiento uniforme [5] significa que intervalos de igual amplitud contenidos en el intervalo $[a,b]$ tienen la misma probabilidad de ocurrir.

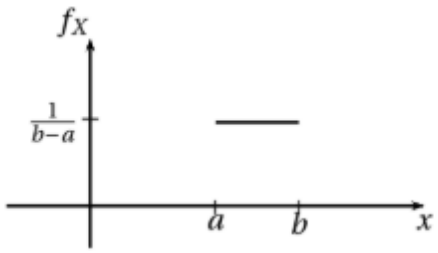

Figura 1: Gráfica de la distribución uniforme en el intervalo $[a,b]$

La función de la distribución acumulativa $F(x)$, para una variable aleatoria X uniformemente distribuida, se puede representar por:

$$F(x) = \int_a^x \frac{1}{b-a} dt \;=\; \frac{x-a}{b-a} \qquad 0 \leq F(x) \leq 1. \tag{7}$$

El valor esperado y la varianza de una variable aleatoria uniformemente distribuida están dados por las siguientes expresiones:

$$E[X] = \int_a^b \frac{1}{b-a} x \, dx \;=\; \frac{b+a}{2} \tag{8}$$

$$V[X] = \int_a^b \frac{(x-E[X])^2}{b-a} dx \;=\; \frac{(b-a)^2}{12} \tag{9}$$

Para simular una distribución uniforme sobre cierto intervalo conocido $[a,b]$ deberemos, en primer lugar, obtener la transformación inversa para la ecuación (5) de acuerdo con la ecuación (2). El procedimiento es el siguiente:

$$\begin{aligned} r &= F(x) \\ r &= \frac{x-a}{b-a} \\ r(b-a) &= x-a \\ r(b-a)+a &= x \end{aligned}$$

Entonces

$$x = a + r(b-a) \qquad 0 \leq r \leq 1. \tag{10}$$

donde a y b son los parámetros de la distribución uniforme y el valor r será el número obtenido de nuestro generador pseudoaleatorio. Así, para una cantidad predefinida de iteraciones, podemos obtener una lista de longitud n de números distribuidos uniformemente. El código Python necesario para generar dicha lista como resultado de llamar a la función iteradas veces proveyendo los parámetros a y b se muestra a continuación.

```
from numpy.random import random

# Uniform Distribution Generator
def uniform(a, b):
    r = random()
    x = a + (b - a) * r
    return x

# Main. Parameters can be modified down here
iterations = 500
a = 5
b = 20
uniform_values = np.zeros(iterations)
for i in range(iterations):
    uniform_values[i] = uniform(a, b)
print(uniform_values)
```

3.1.2. Distribución Normal

Decimos que una variable aleatoria X tiene una distribución normal [5] de parámetros μ y σ, donde $\sigma > 0$, y notamos $X \sim \mathcal{N}(\mu, \sigma)$ cuando su función de densidad de probabilidad es:

$$f_x(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{11}$$

La distribución normal es la que con más frecuencia aparece en estadística y en la teoría de probabilidades. La gráfica de su función de densidad tiene una forma acampanada y es simétrica respecto de un determinado parámetro estadístico. Esta curva se conoce como campana de Gauss y es el gráfico de una función gaussiana.

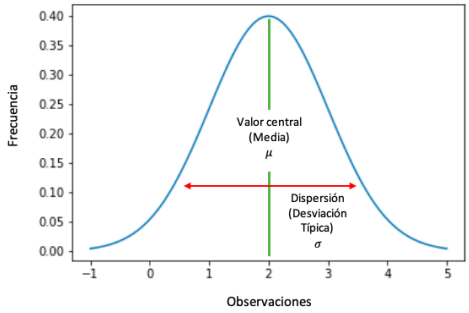

Figura 2: Gráfica de la distribución normal de parámetros μ y σ

La función de la distribución acumulativa $F(x)$, para una variable aleatoria X normalmente distribuida, se puede representar por:

$$F(x) = \int_{-\infty}^{x} \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right) dt \; = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right] \tag{12}$$

Donde erf es la "función error", y su definición es interesante pero escapa al alcance que pretendemos brindar en la extensión de este trabajo.

El valor esperado y la varianza de una variable aleatoria normalmente distribuida están dados por las siguientes expresiones:

$$E[X] = \mu \tag{13}$$

$$V[X] = \sigma^2 \tag{14}$$

Teorema Central del Límite: aproximando una distribución diferente a distribución normal [5] [11]

Si $X_1, X_2, \cdots, X_n$ son n variables aleatorias independientes cualesquiera, con $E[X_i] = \mu_i$ y $V[X_i] = \sigma_i^2$, entonces, bajo las condiciones de que se asegurará la convergencia si: "las variables que se suman son independientes, idénticamente distribuidas, con valor esperado y varianza finitas"(condiciones válidas en la mayor parte de las aplicaciones) entonces la variable aleatoria $X = X_1 + X_2 + \cdots + X_n$, con n convenientemente grande, tiene una distribución *aproximadamente normal* con parámetros $E[X] = \sum_{i=0}^{n} \mu_i$ y $V[X] = \sum_{i=0}^{n} \sigma_i^2$.

Distribución Normal Estándar [8] Si los parámetros de la distribución normal tienen los valores $\mu_x = 0$ y $\sigma_x = 1$, la función de distribución recibirá el nombre de *distribución normal estándar*, con función de densidad:

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} \tag{15}$$

Cualquier distribución normal se puede convertir a la forma estándar mediante la siguiente substitución:

$$z = \frac{x - \mu_x}{\sigma_x} \tag{16}$$

Para simular una distribución normal utilizaremos el procedimiento denominado *Del Límite Central* y conseguiremos la transformación inversa para la ecuación (9) de acuerdo con la ecuación (2). El procedimiento es el siguiente:

Primero se debe proponer la siguiente interpretación matemática del teorema del límite central: si $r_1, r_2, \cdots, r_S$ representan variables aleatorias independientes, cada una de las cuales posee la misma distribución de probabilidad caracterizada por $E[r_i] = \mu$ y $V[r_i] = \sigma^2$, entonces:

$$\lim_{N\to\infty} P[a < \frac{\sum_{i=1}^{N} ri - N\mu}{\sqrt{N}\sigma} < b] = \frac{1}{\sqrt{2\pi}} \int_a^b e^{-\frac{1}{2}z^2}\, dz \tag{17}$$

Donde:

$$E[\sum_{i=1}^{N} ri] = N\mu \qquad V[\sum_{i=1}^{N} ri] = N\sigma^2 \qquad z = \frac{\sum_{i=1}^{N} ri - N\mu}{\sigma\sqrt{N}} \tag{18}$$

Tanto de la definición de la distribución normal estándar como de la ecuación (14), se sigue que z es un valor de variable aleatoria con distribución normal estándar.

El procedimiento para simular valores normales utilizando computadoras requiere el uso de la suma de K valores de variable aleatoria distribuidos uniformemente; esto es, la suma de $r_1, r_2, \cdots, r_K$, con cada r_i definida en el intervalo $0 < r_1 < 1$. Así, tenemos:

$$\mu = \frac{a+b}{2} = \frac{0+1}{2} = \frac{1}{2} \qquad \sigma = \frac{b-a}{\sqrt{12}} = \frac{1}{\sqrt{12}} \qquad z = \frac{\sum_{i=1}^{K} r_i - \frac{K}{2}}{\sqrt{\frac{K}{12}}} \tag{19}$$

Pero, por definición, z es un valor de variable aleatoria con distribución normal estándar que se puede escribir en la forma sugerida por la ecuación (14), donde x es un valor de variable aleatoria distribuido en forma normal que se va a simular, con media μ_x y varianza σ_x^2. Igualando las ecuaciones (14) y la definición de z en (16), obtenemos:

$$\frac{x - \mu_x}{\sigma_x} = \frac{\sum_{i=1}^{K} r_i - \frac{K}{2}}{\sqrt{\frac{K}{12}}} \tag{20}$$

Y resolviendo para x, tenemos que:

$$x = \sigma_x \sqrt{\frac{K}{12}} (\sum_{i=1}^{K} r_i - \frac{K}{2}) + \mu_x \tag{21}$$

Por lo tanto, mediante la ecuación anterior podemos proporcionar una formulación muy simple para generar valores de variable aleatoria normalmente distribuidos, cuya media será μ_x y la varianza σ_x^2-

Una recomendación es hacer $K = 12$ y se simplifica un poco la fórmula:

$$x = \sigma_x \sqrt{\frac{12}{12}} (\sum_{i=1}^{12} r_i - \frac{12}{2}) + \mu_x \implies x = \sigma_x (\sum_{i=1}^{12} r_i - 6) \tag{22}$$

Así, para una cantidad predefinida de iteraciones, podemos obtener una lista de longitud n de números distribuidos normalmente. El código Python necesario para generar dicha lista como resultado de llamar a la función iteradas veces proveyendo los parámetros μ y σ se muestra a continuación.

```python
from numpy.random import random

# Normal Distribution Generator
def normal(m, d):
    sum = 0
    for _ in range(12):
        r = random()
        sum += r
    x = d * (sum - 6) + m
    return x

# Main. Parameters can be modified down here
iterations = 500
mean = m
deviation = d
normal_values = np.zeros(iterations)
for i in range(iterations):
    normal_values[i] = normal(m, d)
print(normal_values)
```

3.1.3. Distribución Exponencial

Decimos que una variable aleatoria X tiene una distribución exponencial de parámetro $a > 0$ y notamos $X \sim \mathscr{E}(a)$ cuando su función de densidad de probabilidad es

$$f_x(x) = \begin{cases} ae^{-ax} & \text{si } x > 0 \\ 0 & \text{en otro caso} \end{cases} \tag{23}$$

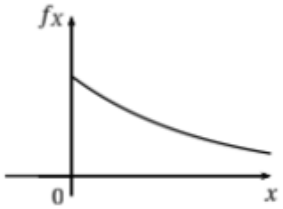

Figura 3: Gráfica de la distribución exponencial

Para los valores de variables aleatorias de tipo exponencial se deben satisfacer las siguientes suposiciones:

1. La probabilidad de que ocurra un evento en el intervalo $[t, (t + \Delta t)]$ es $a\Delta t$
2. a es una constante que no depende de t o de algún otro factor

3. La probabilidad de que durante un intervalo $[t,(t+\Delta t)]$ ocurra más de un evento, tiende a 0 a medida que $\Delta t \to 0$, y su orden de magnitud deberá ser menor que el de $a\Delta t$.

Curiosamente, se ha encontrado que el comportamiento de un considerable numero de procesos dependientes del tiempo satisfacen las anteriores suposiciones un tanto fuertes. Por ejemplo, el intervalo entre los accidentes de una fábrica, la llegada de pedidos a una compañía, el registro de pacientes en un hospital, el aterrizaje de aviones en un aeropuerto, etc. satisfacen una distribución exponencial. Hemos dicho que una variable aleatoria X tiene una distribución exponencial si se puede definir a su función de densidad como: $f(x) = ae^{-ax}$ con $a > 0$ y $x \geq 0$.

Luego la función de distribución acumulativa viene dada por:

$$F(x) = \int_0^x ae^{-ax}dt = 1 - e^{-ax} \tag{24}$$

Consecuentemente $P(X > x) = 1 - (1 - e^{-ax}) = e^{-ax}$. Además, la media y la varianza de X se pueden expresar como:

$$E[X] = \int_0^\infty xae^{-ax}dx = \frac{1}{a} \tag{25}$$

$$V[X] = \int_0^\infty (x - \frac{1}{a})^2 ae^{-ax}dx = \frac{1}{a^2} = (E[X])^2 \tag{26}$$

Para hallar la transformación inversa utilizamos el método que ya hemos explicado, procediendo a resolver $r = e^{-ax}$ para x:

$$\begin{aligned} r &= e^{-ax} \\ \ln(r) &= ln(e^{-ax}) \\ \ln(r) &= -ax \\ x &= -(\frac{1}{a})\ln(r) \end{aligned}$$

Por consiguiente, para cada valor del número pseudoaleatorio r se determina un único valor para x. Los valores de x toman tan sólo magnitudes no negativas debido a que $\ln(r) \leq 0$ para $0 \leq r \leq 1$, y además se ajustan a la función de densidad exponencial con un valor esperado $E[X]$.

Así, para una cantidad predefinida de iteraciones, podemos obtener una lista de longitud n de números distribuidos exponencialmente. El código Python necesario para generar dicha lista como resultado de llamar a la función iteradas veces proveyendo el parámetro a se muestra a continuación.

```python
from numpy.random import random
from numpy import log as ln

# Exponential Distribution Generator
def exponential(ex):
    r = random()
    x = -ex * ln(r)
    return x

# Main. Parameters can be modified down here
iterations = 500
alpha_exp = 3
exponential_values = np.zeros(iterations)
for i in range(iterations):
    exponential_values[i] = exponential(1/alpha_exp)
print(exponential_values)
```

3.1.4. Distribución Gamma

Decimos que una variable aleatoria X tiene una distribución gamma con parámetros $a > 0$, $k > 0$ y notamos $\mathrm{X} \sim \Gamma(a)$ cuando su función de densidad de probabilidad es

$$f_x(x) = \frac{\alpha^k x^{k-1} e^{-\alpha x}}{(k-1)!} \tag{27}$$

Y se considera a x no negativo. Pese a que no existe una forma explícita para describir la función acumulativa de la distribución gamma, *Pearson* [12] ha presentado en forma tabular los valores de la denominada *Función Gamma Incompleta* (más detalles en el texto de Pearson indicado en la fuente). Respecto a la media y la varianza de esta distribución, sus correspondientes expresiones están formuladas como sigue:

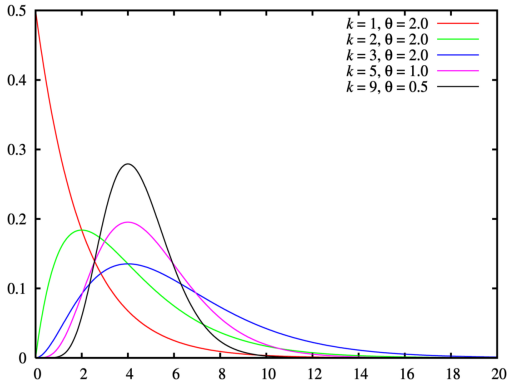

Figura 4: Gráfica de la distribución gamma con diferentes valores para k y para α (en la gráfica es llamado θ)

$$E[X] = \frac{k}{\alpha} \tag{28}$$

$$V[X] = \frac{k}{\alpha^2} \tag{29}$$

Si $K = 1$, entonces la distribución gamma resulta ser idéntica a la distribución exponencial; mientras que si k es un entero positivo, la distribución gamma coincide con la llamada *Distribución de Erlang*. Conviene también anotar que, a medida que k se incrementa, la distribución gamma tiende en forma asintótica, a una distribución normal.

Para generar valores de variable aleatoria con distribución gamma y con una media y varianza dados, se pueden emplear las siguientes fórmulas a fin de determinar los parámetros de $f(x)$ en la ecuación de densidad de la función gamma:

$$\alpha = \frac{E[X]}{V[X]} \qquad k = \frac{E[X]^2}{V[X]} \tag{30}$$

Debido a que para una distribución gamma no se puede formular explícitamente una función de distribución acumulativa, debemos considerar un método alternativo para generar valores de variable aleatoria con distribución gamma. En relación a tales valores que satisfacen la distribución Erlang, éstos se pueden generar con sólo reproducir el proceso aleatorio sobre el cual se basa la distribución de Erlang. Para lograr este resultado se debe tomar la suma de los k valores de la variable aleatoria con distribución exponencial $x_1, x_2, \cdots, x_k$, cuyo valor esperado es el mismo e igual a $\frac{1}{\alpha}$. En consecuencia, el valor de la variable aleatoria (Erlang) x se puede expresar como:

$$x = \sum_{i=1}^{k} x_i = -\frac{1}{\alpha} \sum_{i=1}^{k} \ln r_i \tag{31}$$

O también se puede expresar en una versión equivalente pero computacionalmente más rápida:

$$x = -\frac{1}{\alpha} \ln \prod_{i=1}^{k} r_i \tag{32}$$

Como hemos dicho, la distribución Erlang es equivalente a la función Gamma cuando k toma valores enteros. El problema de generar valores de variable aleatoria cuando el parámetro k no es un entero, queda aún en la actividad como un problema por resolverse, ya que en este caso todavía se formula un método estocástico satisfactorio, sin lo cual no puede justificarse algún proceso de simulación correspondiente.

Sin embargo, puede intentarse lo siguiente [8]:

Si k es un número *racional*, entonces se lo puede expresar mediante la suma de un entero más una fracción, de modo tal que $k = k_i + q$, con $0 < q < 1$; más aún si $k_2 = k_1 + 1$, entonces se tiene que $k_2 - k_1 = 1 - q$. Nótese que aquí sólo consideramos el caso en que $k > 1$. Si k es suficientemente grande, entonces la aproximación lograda con este criterio proporcionará resultados más satisfactorios.

```
from numpy.random import random
from numpy import log as ln

# Gamma Distribution Generator
def gamma(k, alpha):
    tr = 1
    for _ in range(k):
        r = random()
        tr = tr * r
    x = -ln(tr) / alpha
    return x

# Main. Parameters can be modified down here
iterations = 500
k_gamma = 10
alpha_gamma = 3
gamma_values = np.zeros(iterations)
for i in range(iterations):
    gamma_values[i] = gamma(k_gamma, alpha_gamma)
print(gamma_values)
```

Nota: Relacionadas con las variables gamma existe un buen número de distribuciones de probabilidad, de las cuales las más importantes que se pueden mencionar son: la distribución *chi*^{2} (χ^2), y la distribución beta (β).

Para distribución χ^2

Esta distribución es un caso especial de la distribución gamma en donde $\alpha = 0,5$. Por lo tanto se puede utilizar el mismo algoritmo descrito arriba, teniendo como parámetro fijo el α en 0,5; de esta forma queda la media y varianza $E[X] = 2k$ (denominado este valor *grados de libertad*) y $V[X] = 4k$ respectivamente. Se podrá utilizar sin ninguna variación, siempre que k sea un número entero, es decir que los grados de libertad (la media) sea un número par.

Al contrario, si $E[X]$ resulta un número impar, entonces $k = \frac{[X]}{2} - \frac{1}{2}$, y por lo tanto: O también se puede expresar en una versión equivalente pero computacionalmente más rápida:

$$x = -\frac{1}{\alpha} \ln \prod_{i=1}^{k} r_i + z^2 \tag{33}$$

donde z es una variable aleatoria con distribución normal unitaria (en la subsección de distribución normal la explicamos con más detalle). El algoritmo resultante en Python, será básicamente agregar esa variable en las operaciones nada más.

Para distribución β

Esta distribución corresponde a los cocientes que se producen con dos variables gamma x_1 y $(x_1 + x_2)$, donde x_1 y x_2 son 2 variables independientes gamma a las que corresponde un mismo valor de los parámetros α, k_1 y k_2 respectivamente, de modo que de $k = (k_1 + k_2)$ resulte el parámetro de $(x_1 + x_2)$.

La variable beta está dada por:

$$x = \frac{x_1}{x_1 + x_2} \qquad 0 < x < 1 \tag{34}$$

Por lo tanto, para generar un valor de variable aleatoria x con distribución beta, se debe obtener el cociente de dos valores de variable aleatoria con distribución gamma (que cumplan la función anterior), uno con un parámetro k_1 al que llamaremos a, y el otro con un parámetro k_2 al que llamaremos b, y nos servirán para el cálculo de la función de densidad, la media y la varianza.

La función de densidad de la distribución Beta es

$$f_x(x) = \frac{\Gamma(a+b) x^{a-1} (1-x)^{b-1}}{\Gamma(a) + \Gamma(b)} \tag{35}$$

Y posee una media y varianza iguales a:

$$E[X] = \frac{a}{a+b} \qquad V[X] = \frac{E[X]b}{(a+b+1)(a+b)} \tag{36}$$

El algoritmo en Python será una variación del que presentamos para gamma, teniendo en cuenta los detalles de este apartado.

Observación: si este modelo tuviese más de 2 cocientes no constantes, se denomina *Distribución de Dirichlet*.

3.2. Distribuciones de Probabilidad Discretas

3.2.1. Distribución Binomial

Las variables aleatorias definidas por el número de eventos exitosos en una sucesión de n ensayos independientes de Bernoulli[20], para los cuales la probabilidad de éxito es p en cada ensayo, siguen una distribución binomial. Este modelo estocástico también se puede aplicar al proceso de muestreo aleatorio con reemplazo, cuando los elementos muestreados tienen sólo dos tipos de atributos (por ejemplo *si* y *no* o respuestas como *aceptable* o *defectuoso*). El diseño de una muestra aleatoria de n elementos es análoga a n ensayos independientes de Bernoulli, en los que x es un valor binomial que está denotando al número de elementos de una muestra de tamaño n con atributos idénticos. Es ésta la analogía que sitúa la distribución binomial como uno de los modelos más importantes en las áreas de muestreo estadístico y del control de calidad.

La distribución binomial proporciona la probabilidad de que un evento tenga lugar x veces en un conjunto de n ensayos, donde la probabilidad de éxito está dada por p. La función de probabilidad para la distribución binomial se puede expresar de la siguiente forma:

$$f(x) = \binom{n}{x} p^x q^{n-x} \tag{37}$$

donde x se toma como un entero definido en el intervalo finito $0, 1, 2, \cdots, n$, y al que se le asocia el valor $q = (1-p)$ El valor esperado y la varianza de la variable binomial X corresponden a:

$$E[X] = np \tag{38}$$

$$V[X] = npq \tag{39}$$

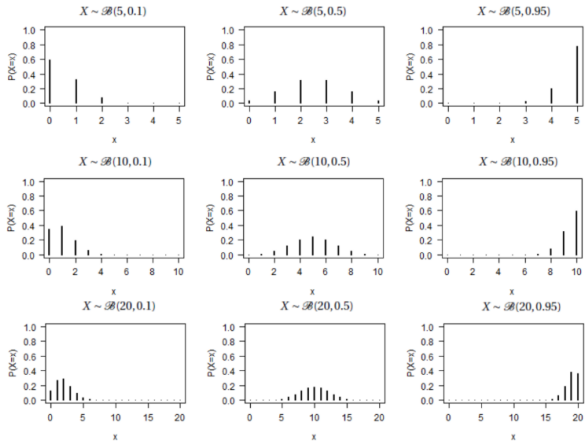

Figura 5: Algunas distribuciones binomiales con diferentes valores para n y p.

Los valores de variable aleatoria con distribución binomial se pueden generar de diversos modos, en el caso de que el valor de n sea moderado resulta ser que el método más eficiente es el basado en la reproducción de ensayos de Bernoulli, siguiendo el método de rechazos. Este método empieza con valores conocidos de p y de n y consiste en generar n

números aleatorios después de fijar x_0 igual a cero. Para cada número aleatorio r_i ($1 \leq i \leq n$) se efectúa una prueba y la variable x_i se incrementa de acuerdo con el siguiente criterio:

$$x_i = x_{i-1} + 1 \qquad si \quad r_i \leq p \tag{40}$$

$$x_i = x_{i-1} \qquad si \quad r_i > p \tag{41}$$

Después de haberse generado n números aleatorios, el valor de x_n será igual al valor de la variable aleatoria con distribución binomial x. Este procedimiento se puede repetir tantas veces como valores binomiales se requieran. El código Python necesario para generar dicha lista como resultado de llamar a la función iteradas veces proveyendo los parámetros n y p se muestra a continuación.

```python
from numpy.random import random

# Binomial Distribution Generator
def binomial(n, p):
    x = 0
    for _ in range(1, n):
        r = random()
        if (r - p <= 0):
            x += 1
    return x

# Main. Parameters can be modified down here
iterations = 500
n = 100
p = 0.5
binomial_values = np.zeros(iterations)
for i in range(iterations):
    binomial_values[i] = binomial(n, p)
print(binomial_values)
```

3.2.2. Distribución Pascal (Binomial Negativa)

La función de distribución de probabilidad para una variable aleatoria x cuando su función posee una distribución binomial negativa (o *Pascal*) está dada por:

$$f(x) = \binom{k+x-1}{x} p^k q^x \qquad x = 0, 1, 2, \cdots \tag{42}$$

Donde p es la probabilidad de éxito y k es el número total de éxitos en una sucesión de $k+x$ ensayos, con x el número o de fallas que ocurren antes de obtener k éxitos. Definimos: $q = 1 - p$, que equivale a la probabilidad de fracaso.

Notamos a la variable aleatoria cuya distribución es binomial negativa como $X \sim \mathscr{BN}(k, p)$

El valor esperado y la varianza de una variable aleatoria con distribución de Pascal están dados por las siguientes expresiones:

$$E[X] = \frac{kq}{p} \tag{43}$$

$$V[X] = \frac{kq}{p^2} \tag{44}$$

Una observación: esa distribución se caracteriza por tener una sobre dispersión, es decir, $V[X] > E[X]$. Sucede exactamente lo mismo con la distribución *Geométrica*, que es solo un caso especial de distribución de Pascal, para cuando $k = 1$.

Para simular una distribución de Pascal deberemos tener en cuenta el siguiente procedimiento:

Para una media y varianza dadas, se pueden determinar los parámetros p y k de la siguiente manera:

$$p = \frac{E[X]}{V[X]} \qquad k = \frac{(E[X])^2}{V[X] - E[X]} \tag{45}$$

Sin embargo, puede suceder que el proceso de simulación se complique considerablemente cuando resulte que en la ecuación anterior el valor que se obtenga al efectuar el cómputo de k no sea entero.

Cuando k es un entero, los valores de la variable aleatoria con distribución de Pascal se pueden generar con sólo considerar la suma de k valores con distribución geométrica ($k = 1$). En consecuencia:

$$x = \frac{(\sum_{i=1}^{k} \ln r_i)}{\ln q} \implies x = \frac{\ln(\prod_{i=1}^{k} r_i)}{\ln q} \tag{46}$$

Viene a ser un valor de variable aleatoria con distribución de Pascal, una vez que su magnitud se redondea con respecto al menor entero más próximo al valor calculado.

Así, para una cantidad predefinida de iteraciones, podemos obtener una lista de longitud n de números distribuidos como binomial negativa. El código Python necesario para generar una lista de números que posean esta distribución es el siguiente:

```
from numpy.random import random
from numpy import floor

# Pascal (Negative Binomial) Distribution Generator
def pascal(k, p):
    tr = 1
    qr = ln(1-p)
    for _ in range(k):
        r = random()
        tr = tr * r
    x = floor(ln(tr)/qr)
    return x

# Main. Parameters can be modified down here
iterations = 500
k_pascal = 4
p_pascal = 0.8
pascal_values = np.zeros(iterations)
for i in range(iterations):
    pascal_values[i] = pascal(k_pascal, p_pascal)
print(pascal_values)
```

3.2.3. Distribución de Poisson

Si tomamos una serie de n ensayos independientes de Bernoulli[20], en cada uno de los cuales se tenga una probabilidad p muy pequeña relativa a la ocurrencia de un cierto evento, a medida que $n \to \infty$, la probabilidad de x ocurrencias está dada por la distribución de Poisson, con parámetro λ y notamos $X \sim \mathscr{P}(\lambda)$

$$f(x) = e^{-\lambda}\frac{\lambda^x}{x!} \qquad x = 0, 1, 2, \ldots \quad \lambda > 0 \tag{47}$$

siempre y cuando permitamos que p se aproxime a cero de manera que se satisfaga la relación $\lambda = np$ consistentemente. De nuestra discusión previa sabemos que np es el valor esperado para la distribución binomial y se puede demostrar que λ es el valor esperado para la distribución de Poisson. De hecho:

$$E[X] = \lambda \tag{48}$$

$$V[X] = \lambda \tag{49}$$

También se puede demostrar que si x es una variable de Poisson con parámetro λ, entonces para valores muy grandes de λ ($\lambda > 10$), se puede utilizar la distribución normal con $E[X] = \lambda$ y $V[X] = \lambda$ para aproximar la distribución de x.

Los eventos que se distribuyen en forma poissoniana ocurren frecuentemente en la naturaleza; por ejemplo, el número de aeroplanos que descienden en un aeropuerto en un período de veinticuatro horas puede ser considerablemente grande. Aún así, resulta muy pequeña la probabilidad de que un avión aterrice durante un segundo determinado. Por lo tanto, podemos esperar que en un período determinado, la probabilidad de que desciendan $0, 1, 2, \ldots$ aviones, obedecerá a las leyes de la distribución de Poisson.

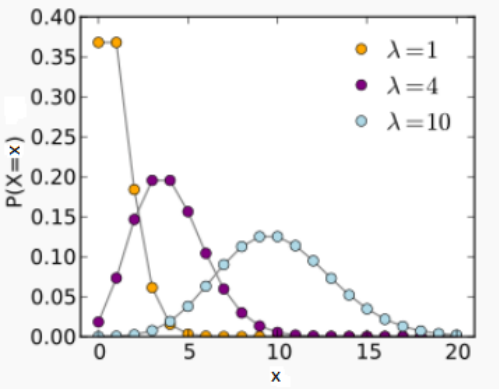

Figura 6: Distribución de Poisson para diferentes valores de λ

Para simular una distribución de Poisson con parámetro λ, nos podemos servir de la relación conocida entre ésta y la distribución exponencial. Se puede justificar que si:

1. El número total de eventos que ocurren durante un intervalo de tiempo dado es independiente del número de eventos que ya han ocurrido previamente al inicio del intervalo
2. La probabilidad de que un evento ocurra en el intervalo de t a $t + \Delta t$ es aproximadamente $\lambda \Delta t$ para todos los valores de t

entonces la función de densidad del intervalo t entre las ocurrencias de eventos consecutivos es $f(t) = \lambda e^{-\lambda t}$, y la probabilidad de que ocurran x eventos durante el tiempo t es:

$$f(x) = e^{-\lambda t} \frac{(\lambda t)^n}{x!} \tag{50}$$

```python
from numpy.random import random
from numpy import exp

# Poisson Distribution Generator
def poisson(L):
    x = 0
    b = exp(-L)
    tr = 1
    while(tr-b >= 0):
        r = random()
        tr = tr * r
        x += 1
    return x

# Main. Parameters can be modified down here
iterations = 500
L = 10 # Lambda
poisson_values = np.zeros(iterations)
for i in range(iterations):
    poisson_values[i] = poisson(L)
print(poisson_values)
```

3.2.4. Distribución Hipergeométrica

Considérese una población que consta de N elementos tales que cada uno de ellos pertenece a la clase I o a la II. Denotemos por Np al número de elementos que pertenecen a la clase I y por Nq al número de elementos que son miembros de la clase II, donde $p+q=1$. Si en una población de N elementos se toma una muestra aleatoria que conste de n elementos ($n < N$) sin que tenga lugar algún remplazo, entonces el número de elementos x de la clase I en la muestra de n elementos, tendrá una distribución de probabilidad hipergeométrica.

Por ejemplo, si el intervalo entre la llegada de órdenes sucesivas de los clientes relativas a cierto producto de la compañía se distribuye geométricamente, por lo tanto la demanda total en cualquier periodo dado de tiempo tendrá una distribución hipergeométrica.

La distribución esta descrita por la siguiente función de probabilidad:

$$f(x) = \frac{\binom{Np}{x}\binom{Nq}{n-x}}{\binom{N}{n}} \qquad 0 \leq x \leq Np; \quad 0 \leq n-x \leq Nq \tag{51}$$

donde x, n y N son enteros. El valor esperado y la varianza se caracterizan como sigue:

$$EX = np \tag{52}$$

$$VX = npq\left(\frac{N-n}{N-1}\right) \tag{53}$$

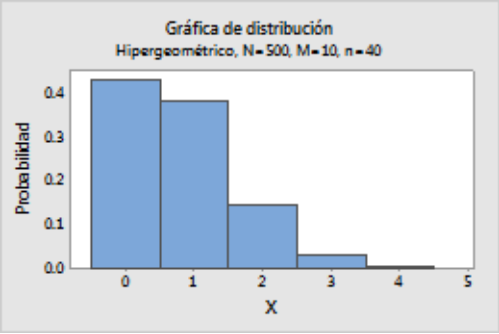

Figura 7: Distribución Hipergeométrica. Aquí Np = M/N = 10/500 = 0.02

La generación de valores hipergeométricos involucra, substancialmente, la simulación de experimentos de muestreo sin reemplazo. En otras palabras, bastara sencillamente con que alteremos el método de ensayos de Bernoulli[20] para generar valores binomiales, con objeto que N y p varíen en forma dependiente respecto al numero total de elementos que previamente se han obtenido entre la población y el numero de elementos de la clase I que se han extraído. A medida que se extrae un elemento de una muestra de n elementos, se reduce el valor de $N = N_0$ de acuerdo con al fórmula:

$$N_i = N_{i-1} - 1 \quad i = 1, 2, \ldots, n. \tag{54}$$

De manera similar, el valor de $p = p_0$ se transforma según

$$p_i = \frac{N_{i-1}p_{i-1} - S}{N_{i-1} - 1} \quad i = 1, 2, \ldots, n. \tag{55}$$

a medida que se saca el i-ésimo elemento de la muestra de n elementos, donde $S = 1$ cuando el elemento de muestra $(i-1)$ pertenece a la clase I y $S = 0$ cuando el elemento de muestra $(i-1)$ pertenece a la clase II. Ciertamente, los valores iniciales de N_0 y P_0 corresponden a:

- N_0: tamaño inicial de la población
- P_0: proporción de la población total que consta de elementos de la clase I

Así, para una cantidad predefinida de iteraciones, podemos obtener una lista de longitud n de números distribuidos hipergeométricamente. El código Python necesario para generar una lista de números que posean esta distribución es el siguiente:

```
from numpy.random import random

# Hypergeometric Distribution Generator
def hypergeometric(N, n, p):
    x = 0
    for i in range(1, n):
        r = random()
        if (r - p <= 0):
            s = 1
            x = x + 1
        else:
            s = 0
        p = (N * p - s) / (N - 1)
        N = N - 1
    return x

# Main. Parameters can be modified down here
iterations = 500
N = 500
n = 80
p = 0.6
hypergeometric_values = np.zeros(iterations)
for i in range(iterations):
    hypergeometric_values[i] = hypergeometric(N, n, p)
print(hypergeometric_values)
```

3.2.5. Distribución Empírica Discreta

En estadística, una función de distribución empírica [13] es la función de distribución asociada con la medida empírica de una muestra.

Ejemplo de una distribución empírica con 5 valores diferentes con una probabilidad de ocurrencia cada uno

$$\begin{cases} 1 = 0{,}22 \\ 2 = 0{,}31 \\ 3 = 0{,}14 \\ 4 = 0{,}23 \\ 5 = 0{,}10 \end{cases} \tag{56}$$

Puede observarse que la suma de todas las probabilidades es igual a 1. Esto también podrá apreciarse en la FDA.

Su función de distribución acumulativa es una *función de paso* que salta $\frac{1}{n}$ en cada uno de los n puntos de datos. Su valor en cualquier valor especificado de la variable medida es la fracción de observaciones de la variable medida que son menores o iguales al valor especificado.

Se puede pensar también a la distribución aleatoria como una generación de valores aleatorios p, $0 \leq p \leq 1$ que representan un valor de probabilidad, y comparar cuál es el mínimo valor de x en la acumulada que sea $x \leq p$.

Función de Distribución Acumulada del ejemplo anterior donde $1 \leq x \leq 5$

$$fda: \begin{pmatrix} 1 = 0{,}22 \\ 2 = 0{,}53 \\ 3 = 0{,}67 \\ 4 = 0{,}90 \\ 5 = 1{,}00 \end{pmatrix}, \text{mínimo valor de } x \text{ tal que } p \leq x \tag{57}$$

Podemos decir que función de distribución empírica es una estimación de la función de distribución acumulativa que generó los puntos en la muestra. Converge con probabilidad 1 a esa distribución subyacente, de acuerdo con el teorema de *Glivenko - Cantelli*, que mencionamos pero no profundizaremos.

A continuación se deja un ejemplo de código en Python que realizamos para representar una distribución empírica que posee 10 valores diferentes de x donde, además de la probabilidad de cada uno de ellos, se almacena el valor correspondiente de la acumulada hasta el mismo.

```
from numpy.random import random

# Empirical Distribution Definition for the Generator
empirical_distribution = {
    0: {"probability": 0.06, "accumulated": 0.06},
    1: {"probability": 0.12, "accumulated": 0.18},
    2: {"probability": 0.08, "accumulated": 0.26},
    3: {"probability": 0.09, "accumulated": 0.35},
    4: {"probability": 0.11, "accumulated": 0.46},
    5: {"probability": 0.16, "accumulated": 0.60},
    6: {"probability": 0.05, "accumulated": 0.65},
    7: {"probability": 0.15, "accumulated": 0.80},
    8: {"probability": 0.16, "accumulated": 0.96},
    9: {"probability": 0.04, "accumulated": 1.00},
}

# Empirical Distribution Analytic Parameters Initialization
def empirical_init():
    mean = sum([key * value["probability"] for (key, value) in
    empirical_distribution.items()])
    variance = sum([( (key - mean)**2 ) * value["probability"] for (key, value) in
     empirical_distribution.items()])
    return mean, variance

# Empirical Distribution Generator
def empirical():
    r = random()
    for (key, value) in empirical_distribution.items():
        if (r <= value["accumulated"]):
            return key

# Main.
iterations = 500 # This parameter can be modified
mean, variance = empirical_init() # Empirical initialization
empirical_values = np.zeros(iterations)
for i in range(iterations):
    empirical_values[i] = empirical()
print(empirical_values)
```

3.3. Tests

Para evaluar si una lista de números aleatorios cumplen realmente una determinada distribución, es necesario hacer *tests* a modo de pruebas de rendimiento, que pueden ser tanto de tipo generalizado para todas las distribuciones, o un test específico de una sola distribución (si alguno de los que se utiliza es de este tipo, se deberá ajustar para poder aplicarse a otras distribuciones o directamente no será posible reutilizarlo).

A continuación definiremos algunos tests accesibles, dejando en claro que hay muchísimos más tests que los ejemplos que daremos.

3.3.1. Tests de Parámetros Estadísticos (Test de Media y Test de Varianza)

Para este test se realiza el cálculo analítico del parámetro a analizar, y luego se compara éste con el resultado de calcular el mismo parámetro pero con los valores obtenidos a partir de la lista de números pseudoaleatorios generados con el algoritmo que escribimos en la simulación para la distribución de probabilidad a analizar.

A partir de un margen de aceptación digamos M (ejemplo, puede ser un 15 %), comparamos si el cociente entre el valor simulado y el valor real está en un intervalo de $1 \pm M$. Si el cociente existe en el intervalo, supera el test para ese parámetro. Caso contrario, no supera el test.

Los parámetros que evaluamos son la media y la varianza. Si bien los evaluamos en 2 momentos independientes, unificamos ambos tests en una sola función sencilla de implementar.

```
import numpy as np

# A list of pseudorandom number passes the mean and deviation tests within an acceptance intervale
acceptance_margin = 0.15
acceptance_interval = [1 - acceptance_margin, 1 + acceptance_margin]

# Simulated vs Analytic Statistic Parameters Calculation - Mean Test and Variance Test
def statistics_parameters_test(numbers_list, real_parameter_result, parameter_name):
    if (parameter_name == "Mean"): simulated_result = np.mean(numbers_list)
    if (parameter_name == "Variance"): simulated_result = np.var(numbers_list)
    print(parameter_name + " value is", simulated_result)
    print("And it is expected to be", real_parameter_result)
    relation = simulated_result / real_parameter_result
    if (acceptance_interval[0] <= relation <= acceptance_interval[1]):
        print(parameter_name + " Test PASSED within the acceptance margin of " + str(acceptance_margin*100) + " %")
    else:
        print(parameter_name + " Test REJECTED within the acceptance margin of " + str(acceptance_margin*100) + " %")
    print()

# Main: calling the test function for mean and variance (example with Exponential Distribution).
statistics_parameters_test(exp_values, 1/alpha_exp, 'Mean')
statistics_parameters_test(exp_values, 1/(alpha_exp**2), 'Variance')
```

3.3.2. Test de comparación gráfica entre FDA anaítica y FDA obtenida en la simulación

Este test es bastante autodescriptivo, lo que hacemos es ordenar los números aleatorios obtenidos de cierta distribución, y luego plotear cada valor representados como un conjunto de puntos en un gráfico (esto representa a la FDA de la distribución que estamos simulando), ploteada junto con la FDA calculada analíticamente. Esto deja una referencia visual concreta de que si los puntos están en concordancia con los valores teóricos.

Ejemplo del algoritmo para el test, con los parámetros de la distribución normal

```
from scipy.stats import norm
from plots import cdf_plots

# Simulated vs Analytic Plot of the cumulative distribution functions
def cdf_comparative_test(numbers_list, distribution_name, loc_p, scale_p):
    sim_x = np.sort(numbers_list)
    sim_y = np.arange(1, len(sim_x)+1) / len(sim_x)
    rv = norm(loc=loc_p, scale=scale_p)
    x = np.linspace(np.maximum(rv.dist.a, min(sim_x)), np.minimum(rv.dist.b, max(sim_x)))
    cdf_plots(x, rv.cdf(x), sim_x, sim_y, distribution_name)

# Main
cdf_comparative_test(normal_values, 'normal', m, d)
```

Nota 1: la función *cdf_plots* es solamente una función ordinaria que grafica los parámetros que le enviamos, de la forma que parezca más conveniente hacerlo.

Nota 2: Para reutilizar este test, basta con importar otra distribución en vez de norm, y reemplazar en la línea donde se define y asigna *RV* (la variable aleatoria).

3.3.3. Test de Kolmogorov Smirnov

Ya hemos mostrado y explicado en el trabajo anterior, el cual puede consultar [7], cómo el test de Kolmogorov Smirnov [14] acepta o rechaza la hipótesis de distribución Uniforme. En este caso nuestra distribución tiene parámetros a y b variables, por lo tanto, para pode reutilizar este test debemos proceder a *normalizar* los valores de la distribución para que se ajusten a $U[0,1]$. Esto lo logramos con un simple algoritmo en donde utilizamos la función *map* en Python:

```
# Normalize function in order to run the Kolmogorov_Smirnov Test
def normalize(n):
  return (n-a) / (b-a)
normalized_values = list(map(normalize, uniform_values))
```

Luego, el código en Python del test es:

```
def test_Kolmogorov_Smirnov(uniform_values):
    print('------------KOLMOGOROV SMIRNOV TEST------------')
    test_array = np.array(uniform_values)
    n = len(test_array)
    test_array.sort()
    # Value below was extracted from table with: alpha = 0.05, n > 50
    d_kolmogorov = 1.36 / (n**0.5)
    Dn_positive = []
    Dn_negative = []
    for i in range(n):
        Dn_positive.append(i/n - test_array[i])
        Dn_negative.append(test_array[i] - (i-1)/n)

    max_dn_pos = max([x for x in Dn_positive])
    max_dn_neg = max([x for x in Dn_negative])

    if (max_dn_pos > max_dn_neg):
        max_general = max_dn_pos
    else:
        max_general = max_dn_neg

    print('is', max_general, ' < ', d_kolmogorov, ' ?')
    if max_general > d_kolmogorov:
        print("Null hypothesis REJECTION, the list of values doesn't correspond to
     an uniform U(a,b) distribution")
        result = "Rejected"
    else:
        print("Null hypothesis ACCEPTATION, the list of values does correspond to
    an uniform U(a,b) distribution")
        result = "Approved"
    print()
```

4. Metodología

Para componer este trabajo práctico, se ha utilizado íntegramente el lenguaje **Python**[15], escrito el código en el IDE **Visual Studio Code**[16] con su correspondiente *Plugin*.

4.1. Librerías y Módulos de Python Utilizados

Numpy [17] Se hizo uso del generador Mersenne-Twister del módulo *random* de esta librería para obtener números aleatorios en una distribución uniforme normalizada $\mathscr{U}(0,1)$. Necesarios en nuestros algoritmos para la generación de números en otras distribuciones diferentes. También se utilizaron funciones varias para manipulación de arrays.

Pyplot [18] Este módulo de la librería **matplotlib** se utilizó para graficar las *Funciones de distribución acumulada* de una distribución obtenida del resultado de la simulación contra la misma función calculada en forma analítica.

Scipy [19] Tomamos el módulo *Stats* que nos ofrece diferentes distribuciones de referencia como exponencial, normal, gamma, etc... que usamos para comparar contra las nuestras generadas en la simulación, y son las que graficamos con *Pyplot* como mencionamos antes.

4.2. Convenciones

A lo largo de este trabajo se consideró al generador uniforme de *numpy* como una fuente sin errores de números pseudoaleatorios, si bien supera los test no se hizo un análisis profundo para ver si el mismo tiene debilidades.

También tomamos a las distribuciones de *scipy* que usamos de referencia para comparar con las nuestras como 'resultados gráficos analíticos' y no se ha realizado por el momento un estudio sobre como esta librería genera los output deseados en su módulo *Stats*.

4.3. Métodos de Resolución Aplicados

Primero se realizó un análisis y luego el correspondiente algoritmo en Python de como generar números pseudoaleatorios en distribuciones **continuas** (uniforme, normal, exponencial, gamma) y **discretas** (binomial, binomial negativa o Pascal, Poisson, hipergeométrica, empírica).

A partir de valores pseudoaleatorios en una distribución uniforme normalizada, mediante ciertas transformaciones que explicamos a detalle con su correspondiente código para cada distribución en la sección teórico-práctica anterior a metodología, obtenemos como resultado un output de otra secuencia de números pseudoaleatorios, pero esta vez corresponden a la nueva distribución deseada.

La cantidad de **números pseudoaleatorios** a generar se piden al usuario para que ingrese por consola el número de pseudoaleatorios deseado a analizar.

Luego sometemos a estos conjuntos de números a varios **tests de calidad**, también explicados con detalle en la sección anterior; realizamos un Test en donde comparamos su valor medio y otro Test con su varianza con respecto a los valore analíticos que deberían tener y si este supera o no el margen de tolerancia especificado. La tolerancia máxima para superar tanto el teste de Media como el test de Varianza fue definido en $\pm 15\,\%$ con respecto al valor analítico real del parámetro correspondiente.

También testeamos a las distribuciones continuas de una forma gráfica, primero ordenando crecientemente sus valores para que adopten la forma de la función de distribución acumulada y luego plotteándolos en una gráfica de puntos sobre la curva analítica real de la distribución. Si bien este test no nos aporta un valor numérico puesto que no se ha realizado un análisis de correlación mediante regresión polinómica, se ha optado por dejar el resultado gráfico ya que por sí mismo brinda información visual interesante con respecto a la simulación y el cálculo analítico.

También testeamos al generador de números aleatorios de distribución uniforme estandarizada mediante el test de *Kolmogorov Smirnov*.

Por último, recolectamos todos los valores numéricos y gráficos obtenidos de los tests e hicimos una tabla resumiendo todos los resultados.

5. Resultados

Los resultados que se mostrarán a continuación fueron obtenidos en una corrida del programa que realizamos en Python, con conjuntos formados por 500 números pseudoaleatorios obtenidos cada uno de ellos a través de cada generador de distribución de probabilidad.

5.1. Resultados de los Tests: Resumen

Distribución de probabilidad	Test de Media	Test de Varianza	Test Gráficas FDA
UNIFORME	PASSED	PASSED	PASSED
NORMAL	PASSED	PASSED	PASSED
EXPONENCIAL	PASSED	PASSED	PASSED
GAMMA	PASSED	PASSED	PASSED
BINOMIAL	PASSED	PASSED	-
PASCAL	REJECTED	REJECTED	-
POISSON	PASSED	PASSED	-
HIPERGEOMÉTRICA	PASSED	PASSED	-
EMPÍRICA	PASSED	PASSED	-

5.2. Test de Media y Varianza

Margen de tolerancia: $\pm 15\,\%$

Distribución Uniforme

Parámetro	Valor esperado	Valor obtenido	Resultado
MEDIA	14.0	13.941	PASSED
VARIANZA	5.333	5.223	PASSED

Distribución Normal

Parámetro	Valor esperado	Valor obtenido	Resultado
MEDIA	10.0	9.928	PASSED
VARIANZA	4.0	4.11	PASSED

Distribución Exponencial

Parámetro	Valor esperado	Valor obtenido	Resultado
MEDIA	0.333	0.324	PASSED
VARIANZA	0.111	0.104	PASSED

Distribución Gamma

Parámetro	Valor esperado	Valor obtenido	Resultado
MEDIA	0.333	0.356	PASSED
VARIANZA	0.111	0.114	PASSED

Distribución Binomial

Parámetro	Valor esperado	Valor obtenido	Resultado
MEDIA	50.0	49.62	PASSED
VARIANZA	25.0	27.067	PASSED

Distribución de Pascal (Binomial Negativa)

Parámetro	Valor esperado	Valor obtenido	Resultado
MEDIA	1.0	1.982	REJECTED
VARIANZA	1.25	1.72	REJECTED

Distribución de Poisson

Parámetro	Valor esperado	Valor obtenido	Resultado
MEDIA	10	11.07	PASSED
VARIANZA	10	9.577	PASSED

Distribución Hipergeométrica

Parámetro	Valor esperado	Valor obtenido	Resultado
MEDIA	48.0	47.414	PASSED
VARIANZA	16.160	15.811	PASSED

Distribución Empírica

Parámetro	Valor esperado	Valor obtenido	Resultado
MEDIA	4.78	4.768	PASSED
VARIANZA	7.249	7.434	PASSED

5.3. Test de Kolmogorov Smirnov

El valor obtenido debe ser **menor** al esperado

Generador de distribución de probabilidad	Valor esperado	Valor obtenido	Resultado
UNIFORME (normalizada)	0.033	0.060	PASSED

5.4. Test de FDA: Gráficas obtenidas

Se muestra a continuación para cuatro distribuciones continuas:

- Distribución uniforme
- Distribución normal
- Distribución exponencial
- Distribución gamma

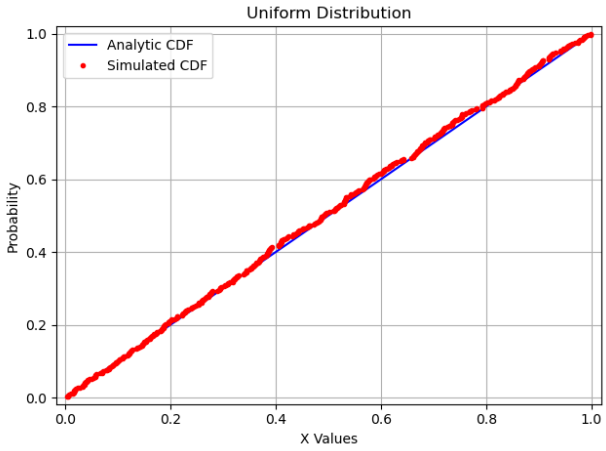

Figura 8: Distribución Uniforme - Simulada vs Analítica

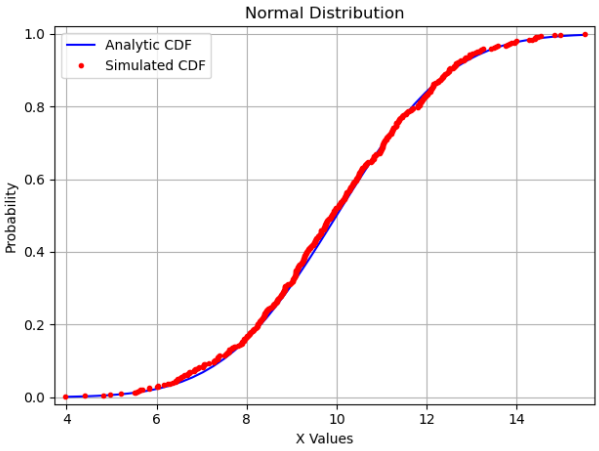

Figura 9: Distribución Normal - Simulada vs Analítica

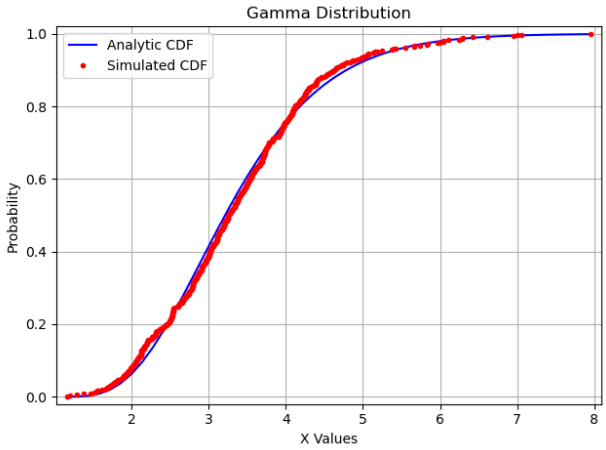

Figura 10: Distribución Gamma - Simulada vs Analítica

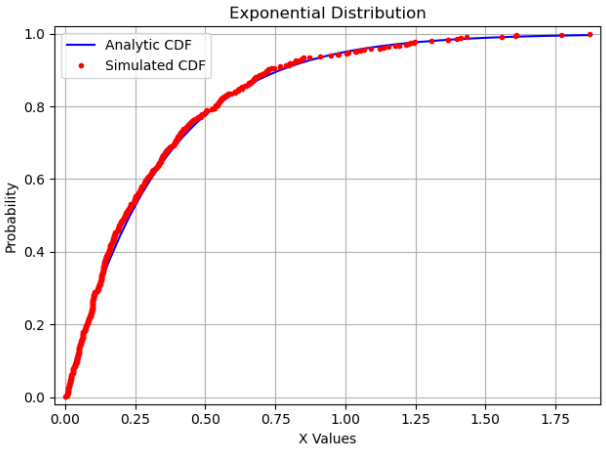

Figura 11: Distribución Exponencial - Simulada vs Analítica

6. Discusión

6.1. Interpretación de los resultados

Los resultados gráficos nos muestran que los valores simulados para las cuatro distribuciones descritas anteriormente tienden prácticamente a los valores esperados de cada una de ellas. Nuestro número de iteraciones (n) fue de 500, creemos que es un buen número y que en caso de aumentarlo, cada vez tendería más a ser coincidente con la gráfica esperada.

En cuanto a los test, creemos que un 15 % es un buen margen de aceptación para determinar si los parámetros estadísticos se están cumpliendo y consecuentemente, el generador de distribuciones de probabilidad está funcionando correctamente. Como podemos ver, para todos nuestros generadores ambas pruebas (Media y Varianza) los test han sido positivos, aunque exceptuando el generador de distribución binomial negativa (o Pascal). Esto nos da una idea de que efectivamente están funcionando como deberían aún definiendo nosotros los parámetros que recibe cada una de las distintas distribuciones, incluso jugando con la variación de los mismos.

Todos los algoritmos para generar números aleatorios en diferentes distribuciones que hemos explicitado en este informe están basados en la utilización de un generador uniforme (ya que utilizamos el *Mersenne-Twister* de la librería de Python *Numpy* [17]) y luego aplicar transformaciones y procesar los resultados para que estos valores pasen a tener correctamente otra distribución de probabilidad. Por lo tanto, si este generador supera con éxito las pruebas de aleatoriedad para uniforme, por extensión los números generados en otras distribuciones basados en él permanecen pseudoaleatorios.

Nota: siempre y cuando no se utilicen operaciones que puedan 'romper' la aleatoriedad, como llevar todos los números a una multiplicación por cero.

6.2. Limitaciones

En cuanto a las gráficas, nos hemos limitado a algunas de las distribuciones continuas puesto que no pudimos encontrar documentación oficial de Scipy que nos permitiera graficar las distribuciones discretas de una manera sencilla o bien los parámetros de escalado no estaban del todo explícitos.

También hicimos uso de un solo generador de números aleatorios con un solo test (K-S) para probar la eficacia del mismo.

Con respecto a la binomial negativa, no encontramos un algoritmo que realmente pase las pruebas como las pasan las otras distribuciones por el momento.

6.3. Recomendaciones futuras

Este trabajo puede retomarse y agregar gráficas para distribuciones discretas, así como también tener en cuenta otras gráficas tales como *histogramas* o *heatmaps*.

También se podrían realizar otros tests tanto para el generador, como también para testear individualmente las distribuciones. Ya sea generalizando el test de K-S, como haciendo otros test por ejemplo el de χ^2.

Puede ampliarse sin duda, teniendo en cuenta más distribuciones que las que utilizamos y otros métodos diferentes; así como también tener en cuenta la complejidad y el rendimiento que posee cada método.

7. Conclusiones

Como se puede observar, es posible generar distintas distribuciones aleatorias de probabilidad siguiendo algoritmos sencillos y rápidos de implementar. Hemos demostrado que, además, cumplen con las pruebas a las que fueron sometidos. Estos generadores que hemos presentado son contundentes y confiables, se deja libertad al lector para utilizarlo en programas más complejos o aplicarlos en situaciones que lo requiera.

En síntesis, dados los positivos resultados de las pruebas, consideramos que los algoritmos para generar distribuciones aleatorias que hemos presentado pueden llegar a ser utilizados en diferentes modelos de simulación en los que se necesiten diferentes distribuciones, aquellas que se adapten mejor a la realidad del problema en cuestión.

Referencias

[1] Random.org
https://www.random.org/analysis/

[2] Wikipedia EN. Hardware random number generator.
https://en.wikipedia.org/wiki/Hardware_random_number_generator

[3] Github IO. Numeros Pseudoaleatorios.
https://tereom.github.io/est-computacional-2018/numeros-pseudoaleatorios.html

[4] Wikipedia EN. Mersenne-Twister.
https://en.wikipedia.org/wiki/Mersenne_Twister

[5] Raúl Katz - Pablo Sabatinelli (2018). Probabilidad y Estadística: Variables Aleatorias Discretas y algunas Distribuciones de Probabilidad

[6] UTN, Apunte Cátedra Simulación.
http://hemaruce.angelfire.com/Unidad_III.pdf

[7] Joshua Acciarri - Nicolás Antonelli (2020).
TP2.1 - Generadores Pseudoaleatorios, UTN Argentina Facultad Regional Rosario

[8] Thomas Naylor
Técnicas de Simulación en Computadoras

[9] Wikipedia ES. Función de Distribución Acumulada (FDA).
https://es.wikipedia.org/wiki/Función_de_distribución

[10] Wikipedia ES. Función de Densidad de Probabilidad (FDP).
https://es.wikipedia.org/wiki/Función_de_densidad_de_probabilidad

[11] Wikipedia ES. Teorema Central del Límite.
https://es.wikipedia.org/wiki/Teorema_del_límite_central

[12] Pearson, 1922. Tables of the Incomplete Gamma-Function.
https://www.amazon.es/Tables-Incomplete-Gamma-Function-Karl-Pearson/dp/5518643063

[13] Wikipedia EN. Empirical Distribution Function.
https://en.wikipedia.org/wiki/Empirical_distribution_function

[14] Info importante sobre pruebas de aleatoriedad. Estadística Computacional, Antonio Salmeon Cerdán y María Morales Giraldo, 2001

[15] Python Doc. Python 3.8.2 | Documentación Oficial.
https://docs.python.org/3/

[16] Microsoft (VSC). Visual Studio Code Official Webpage
https://code.visualstudio.com/

[17] Numpy Doc. Numpy 1.18 | Documentación Oficial.
https://numpy.org/doc/1.18/

[18] Pyplot Doc. Pyplot 3.1.1 | Documentación Oficial.
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.html

[19] SciPy Doc. SciPy Library | Documentación Oficial.
https://www.scipy.org/scipylib/index.html

[20] Wikipedia ES Ensayo de Bernoulli.
https://es.wikipedia.org/wiki/Ensayo_de_Bernoulli