TP 2.2 - GENERADORES DE NÚMEROS PSEUDOALEATORIOS DE DISTINTAS DISTRIBUCIONES DE PROBABILIDAD.

Santino Cataldi
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
cataldisantinonanu@gmail.com

Marcos Godoy
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
mjgodoy2002@gmail.com

Matias Luhmann
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
luhmannm0@gmail.com

Matias Marquez
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
matiastmarquez@gmail.com

Tomás Wardoloff
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
tomaswardoloff@gmail.com

20 de mayo de 2025

ABSTRACT

Este trabajo se centra en el desarrollo e implementación de generadores de números pseudoaleatorios para nueve distribuciones de probabilidad fundamentales, tanto continuas (Uniforme, Exponencial, Gamma, Normal) como discretas (Pascal, Binomial, Hipergeométrica, Poisson, Empírica). Se aplicaron los métodos de la transformada inversa y de rechazo en Python 3.x, basándose en el análisis teórico de cada distribución, incluyendo sus funciones de densidad/probabilidad y parámetros. La validación de los generadores se efectuó mediante la comparación visual de histogramas (N=5000 muestras) con las distribuciones teóricas, demostrando una sólida concordancia y proveyendo herramientas robustas para la simulación de sistemas.

1. Introducción

El presente trabajo práctico tiene como objetivo realizar un estudio detallado de diversas distribuciones de probabilidad, tanto continuas como discretas. Para cada distribución, se abordará su fundamentación teórica, incluyendo parámetros, función de probabilidad o densidad, y representaciones gráficas. Además, se desarrollará el procedimiento matemático para obtener la función de distribución acumulada (FDA) y su inversa, cuando sea aplicable para el método de la transformada inversa.

Se implementarán generadores de variables aleatorias para cada distribución en Python 3.x, utilizando el método de la transformada inversa o el método de rechazo según corresponda. Finalmente, se realizará un testeo de los valores generados, comparándolos con las distribuciones teóricas mediante histogramas y gráficos de función de probabilidad/densidad.

El objetivo final es consolidar la comprensión de estas distribuciones y los métodos para generar muestras de las mismas, herramientas fundamentales en el campo de la simulación de sistemas.

2. Descripción del Trabajo

El trabajo consiste en hacer un estudio de las distintas distribuciones de probabilidad, que aunque no son todas las que existen, las mismas abarcan una gran cantidad de los fenómenos de interés para la simulación. De cada una se deben realizar distintas tareas como se detalla a continuación:

- Introducir la distribución de probabilidad en forma teórica; se deben incluir su/s parámetro/s de entrada, función de probabilidad, gráficas, etc.
- Describir matemáticamente el paso a paso para la obtención de su función acumulada e inversa.
- Elaborar un programa por cada distribución de probabilidad en lenguaje Python 3.x.
- Testear la generación de valores de la forma más conveniente para cada caso.

La siguiente tabla resume las distribuciones a estudiar y las tareas asociadas:

Cuadro 1: Listado de distribuciones y tareas a realizar con cada una.

Distribución de Probabilidad	**Tipo**	**T. Inversa**	**M. Rechazo**	**Código**	**Testeo**
UNIFORME	continua	si	si	si	si
EXPONENCIAL	continua	si	si	no	si
GAMMA	continua	no	si	no	no
NORMAL	continua	si	si	si	si
PASCAL	discreta	no	si	no	no
BINOMIAL	discreta	no	si	no	si
HIPERGEOMÉTRICA	discreta	no	si	no	no
POISSON	discreta	no	si	no	si
EMPÍRICA DISCRETA	discreta	no	si	no	si

Nota: La tabla original del TP indica "no"para código y testeo en algunas distribuciones implementadas por rechazo. Sin embargo, el código Python provisto sí las implementa y testea. Este informe seguirá las implementaciones del código provisto.

3. Desarrollo de las Distribuciones

3.1. Distribución Uniforme (Continua)

3.1.1. Introducción Teórica

Una variable aleatoria continua X sigue una distribución uniforme en el intervalo $[a, b]$, denotada como $X \sim U(a, b)$, si su función de densidad de probabilidad (FDP) es constante dentro de este intervalo y cero fuera de él.

- **Parámetros de entrada:**
  - a: Límite inferior del intervalo ($a \in \mathbb{R}$).
  - b: Límite superior del intervalo ($b \in \mathbb{R}$, con $b > a$).
- **Función de Densidad de Probabilidad (FDP):**

$$f(x; a, b) = \begin{cases} \frac{1}{b-a} & \text{si } a \leq x \leq b \\ 0 & \text{en otro caso} \end{cases} \tag{1}$$

- **Media:** $E[X] = \frac{a+b}{2}$
- **Varianza:** $Var(X) = \frac{(b-a)^2}{12}$

La forma típica de la FDP es un rectángulo.

3.1.2. Función Acumulada (FDA) e Inversa

La Función de Distribución Acumulada $F(x)$ se define como $P(X \leq x)$.

- Para $x < a$: $F(x) = \int_{-\infty}^{x} f(t)dt = \int_{-\infty}^{x} 0dt = 0$.
- Para $a \leq x \leq b$: $F(x) = \int_{-\infty}^{a} 0dt + \int_{a}^{x} \frac{1}{b-a}dt = \left[\frac{t}{b-a}\right]_{a}^{x} = \frac{x-a}{b-a}$.
- Para $x > b$: $F(x) = \int_{-\infty}^{a} 0dt + \int_{a}^{b} \frac{1}{b-a}dt + \int_{b}^{x} 0dt = \frac{b-a}{b-a} = 1$.

Resumiendo la FDA:

$$F(x; a, b) = \begin{cases} 0 & \text{si } x < a \\ \frac{x-a}{b-a} & \text{si } a \leq x \leq b \\ 1 & \text{si } x > b \end{cases} \tag{2}$$

Para obtener la inversa de la FDA, $F^{-1}(u)$, igualamos $F(x) = u$ para $0 \leq u \leq 1$ (que corresponde a $a \leq x \leq b$): $u = \frac{x-a}{b-a}$ Despejando x: $u(b-a) = x - a\ x = a + u(b-a)$ Por lo tanto, la función inversa es:

$$X = F^{-1}(U) = a + (b-a)U \tag{3}$$

donde U es una variable aleatoria con distribución uniforme $U(0, 1)$.

3.1.3. Código Python (Generador)

El siguiente código implementa la generación de variables aleatorias uniformes utilizando el método de la transformada inversa.

```python
# 1. UNIFORME (continua) - Método: Transformada Inversa
def generar_uniforme(a, b):
    if a >= b: raise ValueError("El parámetro 'a' debe ser menor que 'b'.")
    u = generar_U01()
    return a + (b - a) * u
```

Listing 1: Generador de la Distribución Uniforme en Python.

Nota: Se asume la existencia de una función `random.random()` que genera $U(0, 1)$.

3.1.4. Testeo

Se generaron $N = 5000$ muestras de una distribución Uniforme con parámetros $a = 2$ y $b = 10$. La Figura 1 muestra el histograma de las muestras generadas superpuesto con la FDP teórica.

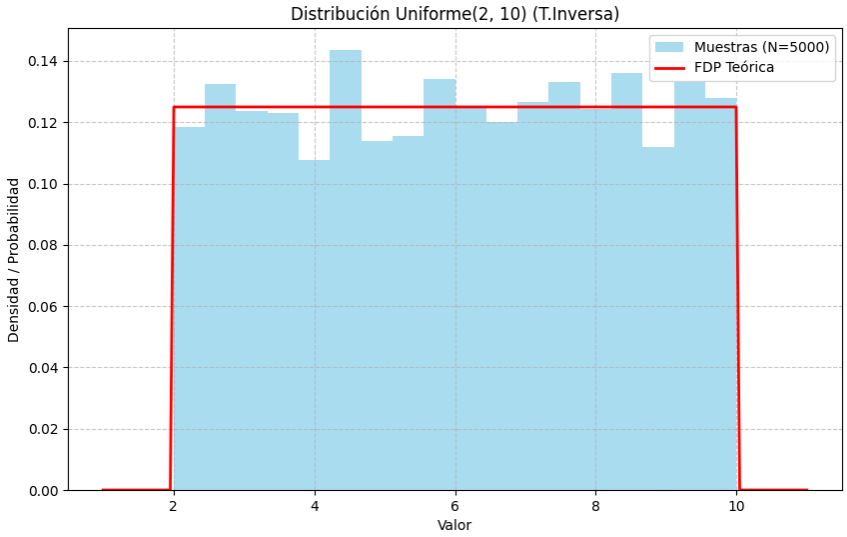

Figura 1: Comparación del histograma de muestras generadas para $U(2, 10)$ (Transformada Inversa) y su FDP teórica.

Como se observa, el histograma de las muestras generadas se aproxima adecuadamente a la FDP teórica, validando la correcta implementación del generador.

3.2. Distribución Exponencial (Continua)

3.2.1. Introducción Teórica

La distribución exponencial describe el tiempo entre eventos en un proceso de Poisson. Se denota $X \sim Exp(\lambda)$.

- **Parámetro de entrada:** λ: Tasa de ocurrencia ($\lambda > 0$).
- **Función de Densidad de Probabilidad (FDP):**

$$f(x; \lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases} \tag{4}$$

- **Media:** $E[X] = \frac{1}{\lambda}$
- **Varianza:** $Var(X) = \frac{1}{\lambda^2}$

3.2.2. Función Acumulada (FDA) e Inversa

Para $x \geq 0$: $F(x) = 1 - e^{-\lambda x}$.

$$F(x; \lambda) = \begin{cases} 1 - e^{-\lambda x} & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases} \tag{5}$$

La función inversa es:

$$X = F^{-1}(U) = -\frac{\ln(U)}{\lambda} \tag{6}$$

donde $U \sim U(0, 1)$.

3.2.3. Código Python (Generador)

```
# 2. EXPONENCIAL (continua) - Método: Transformada Inversa
def generar_exponencial(lam):
    if lam <= 0: raise ValueError("El parámetro 'lam' (lambda) debe ser positivo."
    )
    u = generar_U01()
```

```
    if u == 0: return float('inf')
    return -math.log(u) / lam
```

Listing 2: Generador de la Distribución Exponencial en Python.

3.2.4. Testeo

Se generaron $N = 5000$ muestras de $Exp(0{,}5)$. La Figura 2 compara el histograma con la FDP teórica.

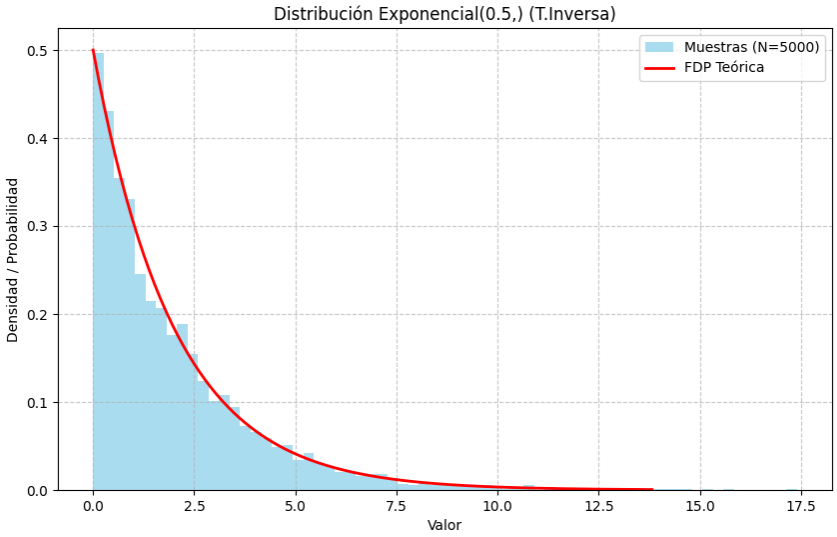

Figura 2: Comparación del histograma de muestras generadas para $Exp(0{,}5)$ (Transformada Inversa) y su FDP teórica.

El histograma concuerda con la FDP teórica.

3.3. Distribución Gamma (Continua)

3.3.1. Introducción Teórica

Se denota $X \sim \Gamma(k, \theta)$.

- **Parámetros de entrada:** k: forma ($k > 0$), θ: escala ($\theta > 0$).
- **Función de Densidad de Probabilidad (FDP):**

$$f(x; k, \theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-x/\theta} \quad \text{para } x > 0 \tag{7}$$

  donde $\Gamma(k)$ es la función Gamma.
- **Media:** $E[X] = k\theta$
- **Varianza:** $Var(X) = k\theta^2$

La forma de la FDP varía significativamente con k.

3.3.2. Función Acumulada (FDA) e Inversa

La FDA no tiene una forma cerrada simple. Se usan métodos de rechazo.

3.3.3. Código Python (Generador - Método de Rechazo)

```python
# 4. GAMMA (continua) - Método: RECHAZO
def generar_gamma_rechazo(k, theta):
    """Genera Gamma(k, theta) usando el método de rechazo de Ahrens-Dieter (GS para k<1, adaptado para k>=1)."""
    if k <= 0 or theta <= 0:
        raise ValueError("k y theta deben ser positivos.")

    # Algoritmo para Gamma(k, 1)
    e_const = math.e
    b_gs = (e_const + k) / e_const
    while True:
        U1 = generar_U01()
        P = b_gs * U1
        if P <= 1:
            X = P ** (1/k)
            U2 = generar_U01()
            if U2 <= math.exp(-X):
                return X * theta # Escalar por theta
        else: # P > 1
            X = -math.log((b_gs - P) / k)
            U2 = generar_U01()
            if U2 <= X ** (k - 1):
                return X * theta # Escalar por theta
```

Listing 3: Generador de la Distribución Gamma en Python (Rechazo).

3.3.4. Testeo

Se generaron $N = 5000$ muestras de $\Gamma(k = 0{,}7, \theta = 2{,}5)$. La Figura 3 compara el histograma con la FDP teórica.

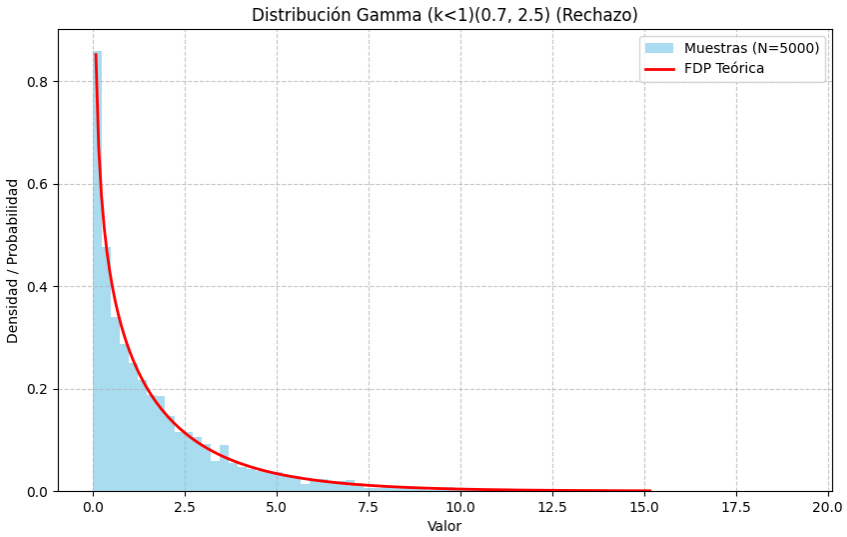

Figura 3: Comparación del histograma de muestras generadas para $\Gamma(0{,}7, 2{,}5)$ (Rechazo) y su FDP teórica.

Los resultados muestran buena concordancia.

3.4. Distribución Normal (Continua)

3.4.1. Introducción Teórica

Se denota $X \sim N(\mu, \sigma^2)$.

- **Parámetros de entrada:** μ: media, σ^2: varianza ($\sigma > 0$).
- **Función de Densidad de Probabilidad (FDP):**

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{8}$$

- **Media:** $E[X] = \mu$
- **Varianza:** $Var(X) = \sigma^2$

FDP simétrica en forma de campana.

3.4.2. Función Acumulada (FDA) e Inversa

La Función de Distribución Acumulada (FDA) de una variable normal estándar $Z \sim N(0, 1)$, denotada como $\Phi(z)$, no tiene una forma cerrada elemental. Sin embargo, su inversa, la función cuantil (o Percent Point Function, PPF), $\Phi^{-1}(u)$, puede ser calculada numéricamente y está disponible en la mayoría de las bibliotecas estadísticas.

Para generar una variable $X \sim N(\mu, \sigma^2)$ usando el método de la transformada inversa con la función cuantil:

1. Generar un número aleatorio $U \sim U(0, 1)$.
2. Calcular $Z_0 = \Phi^{-1}(U)$. Este valor Z_0 sigue una distribución normal estándar $N(0, 1)$.
3. Transformar Z_0 a la distribución deseada: $X = \mu + \sigma Z_0$.

En Python, la función ‘scipy.stats.norm.ppf(u)‘ calcula $\Phi^{-1}(u)$ para la distribución normal estándar.

3.4.3. Código Python (Generador - Transformada Inversa con PPF)

```python
# Es necesario importar la función ppf del módulo stats de scipy
from scipy.stats import norm


# 3. NORMAL (continua) - Método: Transformada Inversa (usando ppf)
def generar_normal(mu, sigma):
    if sigma < 0:
        raise ValueError("Sigma (desviación estándar) debe ser no negativa.")
    if sigma == 0:
        return mu # Devuelve la media si la desviación es cero

    # Generar u ~ U(0,1)

    u = generar_U01()

    # Calcular Z ~ N(0,1) usando la inversa de la FDA (función cuantil o ppf)
    z0 = norm.ppf(u)

    # Transformar a N(mu, sigma^2)
    return mu + sigma * z0
```

Listing 4: Generador de la Distribución Normal en Python (Transformada Inversa con PPF).

3.4.4. Testeo

Se generaron $N = 5000$ muestras de $N(5, 2^2)$. La Figura 4 compara el histograma con la FDP teórica.

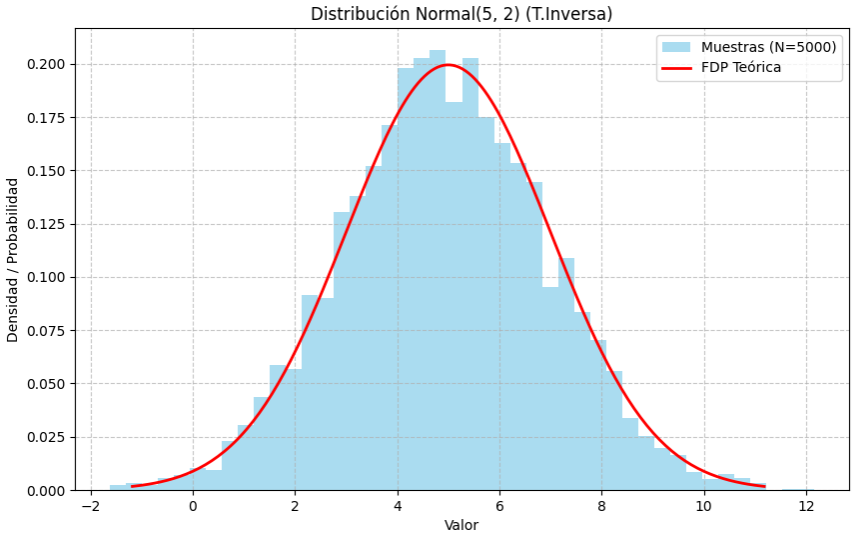

Figura 4: Comparación del histograma de muestras generadas para $N(5, 2^2)$ (Box-Muller) y su FDP teórica.

Se observa buena correspondencia.

3.5. Distribución Pascal (Binomial Negativa) (Discreta)

3.5.1. Introducción Teórica

Modela el número de fracasos k antes de r éxitos, con probabilidad de éxito p.

- **Parámetros:** r: Nro. éxitos ($r > 0$), p: prob. éxito ($0 < p \leq 1$).
- **Función de Probabilidad (FP):**

$$P(K = k; r, p) = \binom{k + r - 1}{k} p^r (1 - p)^k \quad \text{para } k = 0, 1, 2, \ldots \tag{9}$$

- **Media:** $E[K] = \frac{r(1-p)}{p}$
- **Varianza:** $Var(K) = \frac{r(1-p)}{p^2}$

3.5.2. Función Acumulada (FDA) e Inversa

La FDA no tiene forma cerrada simple para su inversa. Se usa método de rechazo.

3.5.3. Código Python (Generador - Método de Rechazo)

```
# 5. PASCAL (discreta) - Método: RECHAZO
def generar_pascal_rechazo(r_exitos, p_exito):
    if not isinstance(r_exitos, int) or r_exitos <= 0: raise ValueError("r_exitos
    entero > 0")
    if not (0 < p_exito <= 1): raise ValueError("p_exito en (0, 1]")
    if p_exito == 1.0: return 0

    media_k = r_exitos * (1 - p_exito) / p_exito
    if r_exitos > 1:
        modo_k = math.floor((r_exitos - 1) * (1 - p_exito) / p_exito)
    else:
        modo_k = 0
    modo_k = max(0, modo_k)
```

```
    max_f_k = pmf_pascal(modo_k, r_exitos, p_exito)

    k_max_estimado = 0
    temp_sum_p = 0
    limit_p = 0.9999
    k_iter = 0
    while temp_sum_p < limit_p and k_iter < (modo_k + 20 * r_exitos + 20):
        pmf_val = pmf_pascal(k_iter, r_exitos, p_exito)
        if pmf_val < 1e-12 and k_iter > modo_k + 5:
            break
        temp_sum_p += pmf_val
        k_iter += 1
    k_max_estimado = k_iter
    k_max_estimado = max(k_max_estimado, modo_k + 5, 10)

    if max_f_k == 0:
        max_f_k_temp = 0
        for k_test in range(k_max_estimado +1):
            current_pmf = pmf_pascal(k_test, r_exitos, p_exito)
            if current_pmf > max_f_k_temp:
                max_f_k_temp = current_pmf
        max_f_k = max_f_k_temp
        if max_f_k == 0: max_f_k = 1e-9

    while True:
        y_candidato = random.randint(0, k_max_estimado)
        u = generar_U01()
        f_y = pmf_pascal(y_candidato, r_exitos, p_exito)
        if u * max_f_k <= f_y:
            return y_candidato
```

Listing 5: Generador de la Distribución Pascal en Python (Rechazo).

3.5.4. Testeo

Se generaron $N = 5000$ muestras de Pascal con $r = 5, p = 0{,}4$. La Figura 5 compara las frecuencias relativas con la FP teórica.

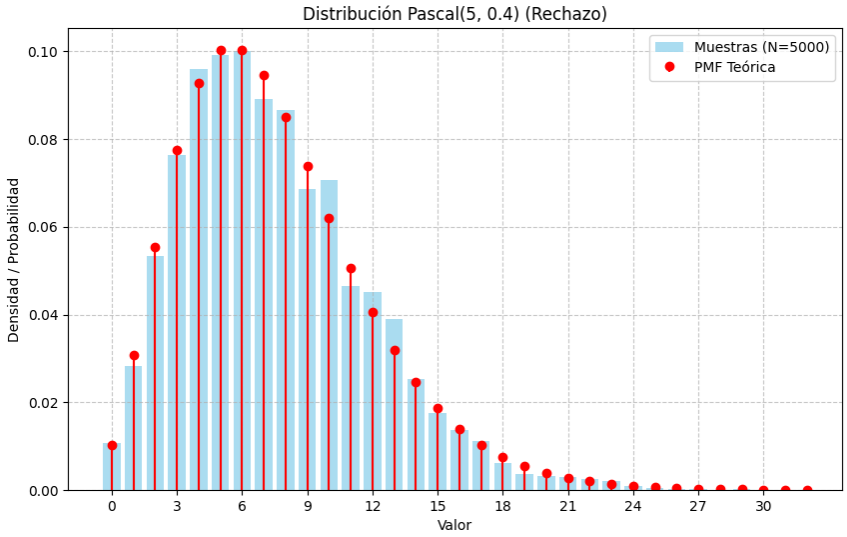

Figura 5: Comparación de frecuencias de muestras generadas para Pascal($r = 5, p = 0{,}4$) (Rechazo) y su FP teórica.

El ajuste es adecuado.

3.6. Distribución Binomial (Discreta)

3.6.1. Introducción Teórica

Modela el número de éxitos k en n ensayos Bernoulli independientes, con probabilidad p. Se denota $X \sim B(n, p)$.

- **Parámetros:** n: Nro. ensayos ($n \geq 0$), p: prob. éxito ($0 \leq p \leq 1$).
- **Función de Probabilidad (FP):**

$$P(K = k; n, p) = \binom{n}{k} p^k (1-p)^{n-k} \quad \text{para } k = 0, \ldots, n \tag{10}$$

- **Media:** $E[K] = np$
- **Varianza:** $Var(K) = np(1-p)$

3.6.2. Función Acumulada (FDA) e Inversa

La FDA no tiene forma cerrada simple para su inversa. Se usa método de rechazo.

3.6.3. Código Python (Generador - Método de Rechazo)

```python
# 6. BINOMIAL (discreta) - Método: RECHAZO
def generar_binomial_rechazo(n_ensayos, p_exito):
    if not isinstance(n_ensayos, int) or n_ensayos < 0: raise ValueError("
    n_ensayos entero >= 0")
    if not (0 <= p_exito <= 1): raise ValueError("p_exito en [0, 1]")
    if n_ensayos == 0: return 0

    modo = math.floor((n_ensayos + 1) * p_exito)
    modo = max(0, min(n_ensayos, modo))
    max_f_k = pmf_binomial(modo, n_ensayos, p_exito)

    if max_f_k == 0:
        if p_exito == 0: return 0
        if p_exito == 1: return n_ensayos
        max_f_k = 1e-9

    while True:
        y_candidato = random.randint(0, n_ensayos)
        u = generar_U01()
        f_y = pmf_binomial(y_candidato, n_ensayos, p_exito)
        if u * max_f_k <= f_y:
            return y_candidato
```

Listing 6: Generador de la Distribución Binomial en Python (Rechazo).

3.6.4. Testeo

Se generaron $N = 5000$ muestras de $B(25, 0{,}25)$. La Figura 6 compara frecuencias relativas con la FP teórica.

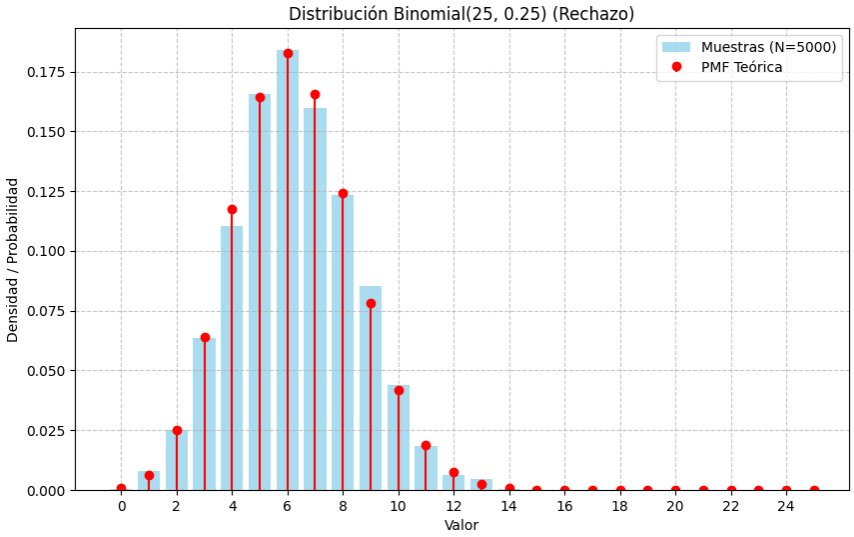

Figura 6: Comparación de frecuencias de muestras generadas para $B(25, 0{,}25)$ (Rechazo) y su FP teórica.

La comparación visual es satisfactoria.

3.7. Distribución Hipergeométrica (Discreta)

3.7.1. Introducción Teórica

Número de éxitos k en muestra de tamaño n sin reemplazo de población N con K éxitos. $X \sim H(N, K, n)$.

- **Parámetros:** N: pob., K: éxitos en pob., n: muestra.
- **Función de Probabilidad (FP):**

$$P(X = k; N, K, n) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}} \tag{11}$$

- **Media:** $E[X] = n\frac{K}{N}$
- **Varianza:** $Var(X) = n\frac{K}{N}(1 - \frac{K}{N})\frac{N-n}{N-1}$

3.7.2. Función Acumulada (FDA) e Inversa

La FDA no tiene forma cerrada simple para su inversa. Se usa método de rechazo.

3.7.3. Código Python (Generador - Método de Rechazo)

```
# 7. HIPERGEOMÉTRICA (discreta) - Método: RECHAZO
def generar_hipergeometrica_rechazo(N_pop, K_ex_pop, n_muestra):
    if not all(isinstance(x, int) for x in [N_pop, K_ex_pop, n_muestra]): raise
    ValueError("Params enteros")
    if not (0 <= K_ex_pop <= N_pop and 0 <= n_muestra <= N_pop): raise ValueError(
    "Params inconsistentes")
    if n_muestra == 0: return 0

    k_min = max(0, n_muestra - (N_pop - K_ex_pop))
    k_max = min(n_muestra, K_ex_pop)

    if k_min > k_max:
```

```
        print(f"Advertencia Hiper: k_min {k_min} > k_max {k_max}. Params: N={N_pop
    }, K={K_ex_pop}, n={n_muestra}")
        return k_min

    modo_aprox = math.floor((n_muestra + 1) * (K_ex_pop + 1) / (N_pop + 2))
    modo = max(k_min, min(k_max, modo_aprox))
    max_f_k = pmf_hipergeometrica(modo, N_pop, K_ex_pop, n_muestra)

    if max_f_k == 0:
        max_f_k_temp = 0
        for k_test in range(k_min, k_max + 1):
            current_pmf = pmf_hipergeometrica(k_test, N_pop, K_ex_pop, n_muestra)
            if current_pmf > max_f_k_temp:
                max_f_k_temp = current_pmf
        max_f_k = max_f_k_temp
        if max_f_k == 0:
            if k_min == k_max and pmf_hipergeometrica(k_min, N_pop,K_ex_pop,
    n_muestra) > 0:
                max_f_k = pmf_hipergeometrica(k_min, N_pop,K_ex_pop,n_muestra)
            else:
                max_f_k = 1e-9

    while True:
        if k_min > k_max :
            return k_min

        y_candidato = random.randint(k_min, k_max)
        u = generar_U01()
        f_y = pmf_hipergeometrica(y_candidato, N_pop, K_ex_pop, n_muestra)
        if u * max_f_k <= f_y:
            return y_candidato
```

Listing 7: Generador de Hipergeométrica en Python (Rechazo).

3.7.4. Testeo

Muestras de $H(60, 15, 20)$. Figura 7 compara frecuencias con FP teórica.

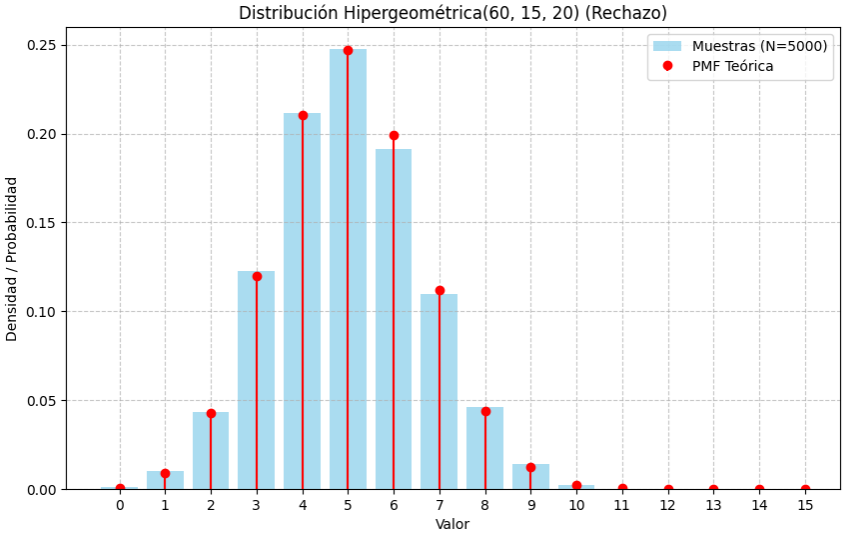

Figura 7: Comparación de frecuencias de muestras generadas para $H(60, 15, 20)$ (Rechazo) y su FP teórica.

El ajuste es adecuado.

3.8. Distribución de Poisson (Discreta)

3.8.1. Introducción Teórica

Número de eventos en intervalo fijo, tasa media λ. $X \sim Pois(\lambda)$.

- **Parámetro:** λ: Nro. medio eventos ($\lambda > 0$).
- **Función de Probabilidad (FP):**

$$P(K = k; \lambda) = \frac{e^{-\lambda}\lambda^k}{k!} \quad \text{para } k = 0, 1, 2, \ldots \tag{12}$$

- **Media:** $E[K] = \lambda$
- **Varianza:** $Var(K) = \lambda$

3.8.2. Función Acumulada (FDA) e Inversa

La FDA no tiene forma cerrada simple para su inversa. Se usa método de rechazo.

3.8.3. Código Python (Generador - Método de Rechazo)

```python
# 8. POISSON (discreta) - Método: RECHAZO
def generar_poisson_rechazo(lam):
    if lam < 0: raise ValueError("lambda >= 0")
    if lam == 0: return 0

    modo = math.floor(lam)
    max_f_k = pmf_poisson(modo, lam)
    if lam > 0 and lam == modo :
         max_f_k = max(max_f_k, pmf_poisson(modo-1, lam))

    k_max_estimado = 0
    temp_sum_p = 0
    limit_p = 0.99999
    k_iter = 0
    iter_limit_k = math.ceil(modo + 10 * math.sqrt(lam) + 20 if lam >0 else 20)

    while temp_sum_p < limit_p and k_iter < iter_limit_k:
        pmf_val = pmf_poisson(k_iter, lam)
        if pmf_val < 1e-12 and k_iter > modo + 5 :
            break
        temp_sum_p += pmf_val
        k_iter += 1
    k_max_estimado = max(k_iter, modo + 5, 5)

    if max_f_k == 0 and lam > 0 :
        max_f_k_temp = 0
        for k_t in range(k_max_estimado +1):
            curr_pmf = pmf_poisson(k_t, lam)
            if curr_pmf > max_f_k_temp: max_f_k_temp = curr_pmf
        max_f_k = max_f_k_temp
        if max_f_k == 0: max_f_k = 1e-9

    while True:
        y_candidato = random.randint(0, k_max_estimado)
        u = generar_U01()
        f_y = pmf_poisson(y_candidato, lam)
        if u * max_f_k <= f_y:
            return y_candidato
```

Listing 8: Generador de Poisson en Python (Rechazo).

3.8.4. Testeo

Muestras de $Pois(8{,}0)$. Figura 8 compara frecuencias con FP teórica.

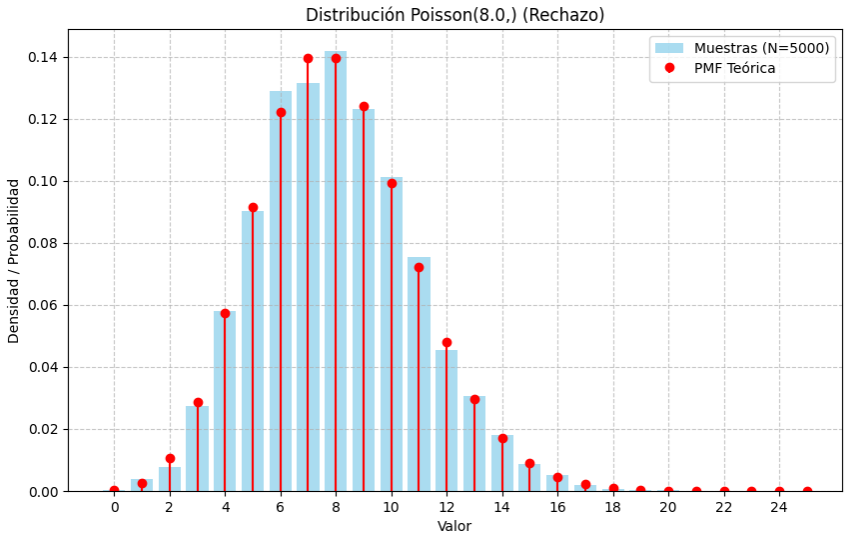

Figura 8: Comparación de frecuencias de muestras generadas para $Pois(8{,}0)$ (Rechazo) y su FP teórica.

La FP de las muestras es consistente con la teórica.

3.9. Distribución Empírica Discreta

3.9.1. Introducción Teórica

Definida por lista de valores $V = \{v_i\}$ y probabilidades $P = \{p_i\}$.

- **Parámetros:** V, P con $\sum p_i = 1$.
- **Función de Probabilidad (FP):** $P(X = x) = p_i$ si $x = v_i$.
- **Media:** $E[X] = \sum v_i p_i$
- **Varianza:** $Var(X) = \sum (v_i - E[X])^2 p_i$

3.9.2. Función Acumulada (FDA) e Inversa

Se usa método de rechazo o transformada inversa con probabilidades acumuladas.

3.9.3. Código Python (Generador - Método de Rechazo)

```python
# 9. EMPÍRICA DISCRETA (discreta) - Método: RECHAZO
def generar_empirica_discreta_rechazo(valores, probabilidades):
    if len(valores) != len(probabilidades): raise ValueError("Longitudes no coinciden")
    if not math.isclose(sum(probabilidades), 1.0, abs_tol=1e-9):
        print(f"Advertencia Empírica: Suma de probs es {sum(probabilidades)}")

    m = len(valores)
    if m == 0: raise ValueError("Listas vacías")

    max_p_i = 0.0
    if any(p > 0 for p in probabilidades):
        max_p_i = max(p for p in probabilidades if p > 0)
```

```
    if max_p_i == 0:
        if m == 1: return valores[0]
        print("Advertencia Empírica: Todas las probabilidades son <= 0.
    Devolviendo el primer valor.")
        return valores[0]

    while True:
        idx_candidato = random.randint(0, m - 1)
        valor_candidato = valores[idx_candidato]
        prob_candidato = probabilidades[idx_candidato]

        u = generar_U01()
        if u * max_p_i <= prob_candidato:
            return valor_candidato
```

Listing 9: Generador de Empírica Discreta en Python (Rechazo).

3.9.4. Testeo

Muestras de Empírica con $V = \{1.,6\}, P = \{0{,}1, 0{,}1, 0{,}3, 0{,}2, 0{,}15, 0{,}15\}$. Figura 9.

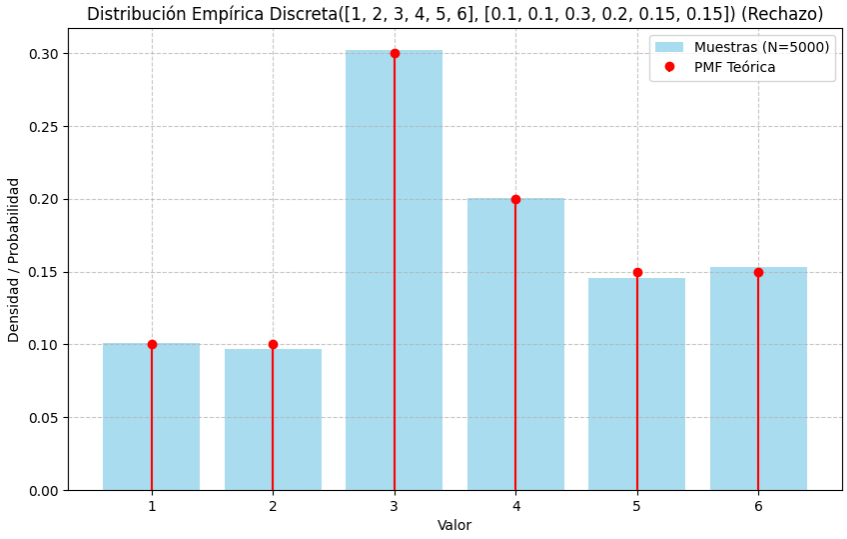

Figura 9: Comparación de frecuencias de muestras generadas para una Distribución Empírica Discreta (Rechazo) y sus probabilidades teóricas.

Las frecuencias se aproximan a las probabilidades teóricas.

4. Conclusiones

En este trabajo se realizó un estudio exhaustivo de nueve distribuciones de probabilidad fundamentales en simulación. Para cada una, se presentó su base teórica, se implementaron generadores de variables aleatorias en Python y se verificó su correcto funcionamiento mediante la comparación visual de las muestras generadas con las funciones de densidad o probabilidad teóricas correspondientes. En todos los casos, se observó una buena concordancia, validando las implementaciones y consolidando la comprensión de estas distribuciones y los métodos para la generación de sus variables aleatorias.

5. Referencias

- Ross, S. M. (2012). *Simulation (5th ed.)*. Academic Press.
- Law, A. M. (2015). *Simulation Modeling and Analysis (5th ed.)*. McGraw-Hill Education.
- Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag.
- Documentación de SciPy: `https://docs.scipy.org/doc/scipy/reference/stats.html`

6. Código Python Completo para Generación y Testeo

A continuación se incluyen todos los scripts de Python empleados en esta investigación, destinados a la generación de variables aleatorias y su posterior análisis gráfico.

config.py

```
N_GLOBAL = 5000
```

base_generator.py

```
import random


def generar_U01():
    return random.random()
```

inverse_transform_generators.py

```
    # inverse_transform_generators.py
import math
from scipy.stats import norm
from base_generator import generar_U01

# --- DISTRIBUCIONES CON TRANSFORMADA INVERSA ---

# 1. UNIFORME (continua) - Método: Transformada Inversa
def generar_uniforme(a, b):
    if a >= b: raise ValueError("El parámetro 'a' debe ser menor que 'b'.")
    u = generar_U01()
    return a + (b - a) * u

# 2. EXPONENCIAL (continua) - Método: Transformada Inversa
def generar_exponencial(lam):
    if lam <= 0: raise ValueError("El parámetro 'lam' (lambda) debe ser positivo."
    )
    u = generar_U01()
    if u == 0: return float('inf') # Teóricamente P(X=inf)=0
    return -math.log(u) / lam

# 3. NORMAL (continua) - Método: Transformada Inversa
def generar_normal(mu, sigma):
    if sigma < 0:
        raise ValueError("Sigma (desviación estándar) debe ser no negativa.")
    if sigma == 0:
        return mu

    u = generar_U01()

    # Calcular Z ~ N(0,1) usando la inversa de la FDA (función cuantil o ppf)
    # Esto es        (u)
    z0 = norm.ppf(u)

    return mu + sigma * z0
```

main.py

```python
    # main.py
import numpy as np # Para pmf_emp y algunos rangos teóricos
from scipy.stats import uniform, expon, gamma, norm, nbinom, binom, hypergeom, poisson # Para FDP/FP teóricas en tests

from config import N_GLOBAL
from inverse_transform_generators import (
    generar_uniforme, generar_exponencial, generar_normal
)
from rejection_method_generators import (
    generar_gamma_rechazo, generar_pascal_rechazo, generar_binomial_rechazo,
    generar_hipergeometrica_rechazo, generar_poisson_rechazo,
    generar_empirica_discreta_rechazo
)
from plotter import testear_distribucion


if __name__ == "__main__":
    # --- Test Uniforme (T. Inversa) ---
    a_unif, b_unif = 2, 10
    testear_distribucion( "Uniforme", generar_uniforme, (a_unif, b_unif),
        lambda x,a,b: uniform.pdf(x, loc=a, scale=b-a),
        N_muestras=N_GLOBAL, es_discreta=False, usa_rechazo=False,
        rango_grafica_teorica=(a_unif -1, b_unif +1))

    # --- Test Exponencial (T. Inversa) ---
    lam_exp = 0.5
    testear_distribucion("Exponencial", generar_exponencial, (lam_exp,),
        lambda x,l: expon.pdf(x, scale=1/l),
        N_muestras=N_GLOBAL, es_discreta=False, usa_rechazo=False,
        rango_grafica_teorica=(0, expon.ppf(0.999, scale=1/lam_exp)))

    # --- Test Normal (T. Inversa - Box Muller) ---
    mu_norm, sigma_norm = 5, 2
    testear_distribucion("Normal", generar_normal, (mu_norm, sigma_norm),
        lambda x,mu,sig: norm.pdf(x, loc=mu, scale=sig),
        N_muestras=N_GLOBAL, es_discreta=False, usa_rechazo=False,
        rango_grafica_teorica=(norm.ppf(0.001, mu_norm, sigma_norm), norm.ppf(0.999, mu_norm, sigma_norm)))

    print("\n--- Iniciando graficación para distribuciones con MÉTODO DE RECHAZO ---")

    # --- Test Gamma (RECHAZO) ---
    k_g, th_g = 0.7, 2.5 # k < 1
    testear_distribucion("Gamma (k<1)", generar_gamma_rechazo, (k_g, th_g),
        lambda x,k,t: gamma.pdf(x, a=k, scale=t),
        N_muestras=N_GLOBAL, es_discreta=False, usa_rechazo=True,
        rango_grafica_teorica=(0, gamma.ppf(0.999, a=k_g, scale=th_g) if k_g > 0 else 10))

    # --- Test Pascal (RECHAZO) ---
    r_pasc, p_pasc = 5, 0.4
    testear_distribucion("Pascal", generar_pascal_rechazo, (r_pasc, p_pasc),
        lambda k,r,p: nbinom.pmf(k, r, p),
        N_muestras=N_GLOBAL, es_discreta=True, usa_rechazo=True,
        rango_grafica_teorica=(0, int(nbinom.ppf(0.999, n=r_pasc, p=p_pasc)) + 5))

    # --- Test Binomial (RECHAZO) ---
    n_bin, p_bin = 25, 0.25
    testear_distribucion("Binomial", generar_binomial_rechazo, (n_bin, p_bin),
        lambda k,n,p: binom.pmf(k, n, p),
        N_muestras=N_GLOBAL, es_discreta=True, usa_rechazo=True,
        rango_grafica_teorica=(0, n_bin))
```

```
61
62      # --- Test Hipergeométrica (RECHAZO) ---
63      N_h, K_h, n_h = 60, 15, 20
64      # La función de SciPy para hipergeométrica usa M (total pop), n (num type I),
    N (sample size)
65      # Nuestro K_ex_pop es n de scipy, N_pop es M de scipy, n_muestra es N de scipy
66      testear_distribucion("Hipergeométrica", generar_hipergeometrica_rechazo, (N_h,
     K_h, n_h),
67          lambda k, M, n, N_sample: hypergeom.pmf(k, M, n, N_sample), # M,n,N ->
    N_pop, K_ex_pop, n_muestra
68          N_muestras=N_GLOBAL, es_discreta=True, usa_rechazo=True,
69          rango_grafica_teorica=(max(0, n_h-(N_h-K_h)), min(n_h, K_h)))
70
71
72      # --- Test Poisson (RECHAZO) ---
73      lam_pois = 8.0
74      testear_distribucion("Poisson", generar_poisson_rechazo, (lam_pois,),
75          lambda k,l: poisson.pmf(k, l),
76          N_muestras=N_GLOBAL, es_discreta=True, usa_rechazo=True,
77          rango_grafica_teorica=(0, int(poisson.ppf(0.9999, lam_pois)) + 5))
78
79      # --- Test Empírica Discreta (RECHAZO) ---
80      val_emp, prob_emp = [1,2,3,4,5,6], [0.1,0.1,0.3,0.2,0.15,0.15]
81      def pmf_emp(k_val, v_list, p_list): # Helper para la teórica de la empírica
82          res = []
83          # Asegurarse que k_val sea iterable, incluso si es un solo número para la
    PMF teórica
84          k_val_iter = np.atleast_1d(k_val)
85          for kv in k_val_iter:
86              try:
87                  idx = v_list.index(kv)
88                  res.append(p_list[idx])
89              except ValueError:
90                  res.append(0)
91          return np.array(res)
92
93      testear_distribucion("Empírica Discreta", generar_empirica_discreta_rechazo, (
    val_emp, prob_emp),
94          lambda k,v,p: pmf_emp(k,v,p),
95          N_muestras=N_GLOBAL, es_discreta=True, usa_rechazo=True,
96          rango_grafica_teorica=(min(val_emp),max(val_emp)))
97
98      print("\n--- Todas las gráficas completadas. Revisa las ventanas emergentes.
    ---")
```

plotter.py

```
1   import matplotlib.pyplot as plt
2   import numpy as np
3   import os
4
5
6   def testear_distribucion(
7       nombre_dist,
8       generador_func,
9       params_dist,
10      scipy_dist_func_pdf_pmf,
11      N_muestras=10000,
12      es_discreta=False,
13      rango_grafica_teorica=None,
14      usa_rechazo=False,
15  ):
16      print(
17          f"\n--- Graficando Distribución: {nombre_dist} con parámetros {params_dist
    } {'(Rechazo)' if usa_rechazo else '(T.Inversa)'} ---"
```

```
    )

    muestras = []
    fallos_generacion_muestra = 0

    for i in range(N_muestras):
        muestra_generada = None
        try:
            muestra_generada = generador_func(*params_dist)
            if muestra_generada is not None:
                muestras.append(muestra_generada)
            else:
                fallos_generacion_muestra += 1
        except NotImplementedError as nie:
            print(f"OMITIDO Gráfico {nombre_dist}: {nie}")
            return
        except Exception as e:
            fallos_generacion_muestra += 1
            if fallos_generacion_muestra > N_muestras / 10 and N_muestras > 100:
                print(
                    f"Demasiados fallos individuales ({fallos_generacion_muestra})
     generando {nombre_dist}. Abortando graficación."
                )
                return
            continue

    if not muestras:
        print(
            f"No se generaron muestras válidas para {nombre_dist}. No se puede
    graficar."
        )
        return

    plt.figure(figsize=(10, 6))
    if es_discreta:
        val_unicos, conteos = np.unique(muestras, return_counts=True)
        frecuencias_relativas = conteos / len(muestras)
        plt.bar(
            val_unicos,
            frecuencias_relativas,
            width=0.8 if len(val_unicos) > 1 else 0.1,
            alpha=0.7,
            label=f"Muestras (N={len(muestras)})",
            color="skyblue",
        )

        if rango_grafica_teorica:
            x_teorico = np.arange(
                rango_grafica_teorica[0], rango_grafica_teorica[1] + 1
            )
        else:
            min_val_obs = min(val_unicos) if len(val_unicos) > 0 else 0
            max_val_obs = max(val_unicos) if len(val_unicos) > 0 else 1
            x_teorico = np.arange(min(min_val_obs, 0), max_val_obs + 1)

        if x_teorico.size == 0:
            print(
                f"  ADVERTENCIA {nombre_dist}: x_teorico está vacío para la PMF te
    órica. Rango: {rango_grafica_teorica}"
            )
        else:
            try:
                y_teorico_pmf = scipy_dist_func_pdf_pmf(x_teorico, *params_dist)
                plt.stem(
                    x_teorico,
```

```
                    y_teorico_pmf,
                    linefmt="r-",
                    markerfmt="ro",
                    basefmt=" ",
                    label="PMF Teórica",
                )

                if len(x_teorico) < 20:
                    plt.xticks(x_teorico)
                elif len(x_teorico) > 0:
                    tick_step = max(1, len(x_teorico) // 10)
                    plt.xticks(
                        x_teorico[::tick_step].astype(int)
                        if np.issubdtype(x_teorico.dtype, np.integer)
                        else x_teorico[::tick_step]
                    )

            except Exception as e_plot_teor:
                print(
                    f"Error graficando PMF teórica para {nombre_dist} {params_dist}: {e_plot_teor}"
                )
                import traceback

                traceback.print_exc()
    else:  # Continua
        plt.hist(
            muestras,
            bins="auto",
            density=True,
            alpha=0.7,
            label=f"Muestras (N={len(muestras)})",
            color="skyblue",
        )
        if rango_grafica_teorica:
            x_teorico = np.linspace(
                rango_grafica_teorica[0], rango_grafica_teorica[1], 200
            )
        else:
            min_val_obs = min(muestras) if len(muestras) > 0 else 0
            max_val_obs = max(muestras) if len(muestras) > 0 else 1
            plot_min = (
                min_val_obs - 0.1 * abs(max_val_obs - min_val_obs)
                if max_val_obs != min_val_obs
                else min_val_obs - 1
            )
            plot_max = (
                max_val_obs + 0.1 * abs(max_val_obs - min_val_obs)
                if max_val_obs != min_val_obs
                else max_val_obs + 1
            )
            x_teorico = np.linspace(plot_min, plot_max, 200)
        try:
            y_teorico_fdp = scipy_dist_func_pdf_pmf(x_teorico, *params_dist)
            plt.plot(x_teorico, y_teorico_fdp, "r-", lw=2, label="FDP Teórica")
        except Exception as e_plot_teor:
            print(
                f"Error graficando FDP teórica para {nombre_dist} {params_dist}: {e_plot_teor}"
            )

    plt.title(
        f"Distribución {nombre_dist}{params_dist} {'(Rechazo)' if usa_rechazo else '(T.Inversa)'}"
    )
```

```
    plt.xlabel("Valor")
    plt.ylabel("Densidad / Probabilidad")
    plt.legend()
    plt.grid(True, linestyle="--", alpha=0.7)
    os.makedirs("graficas", exist_ok=True)
    nombre_archivo = f"graficas/{nombre_dist}_{'_'.join(map(str, params_dist))}_{'rechazo' if usa_rechazo else 'tinversa'}.png"
    plt.savefig(nombre_archivo)
```

probability_functions.py

```
import math
from scipy.special import comb, gammaln  # comb para C(n,k), gammaln para log(Gamma(k))


# Funciones de Densidad/Probabilidad (FDP/FP) necesarias para el método de rechazo
def fdp_gamma(x, k, theta):  # Para Gamma
    if x < 0:
        return 0
    if x == 0:
        if k == 1:
            return 1.0 / theta if theta > 0 else float("inf")  # Exponencial
        if k > 1:
            return 0.0
        if k < 1:
            return float("inf")  # Tiende a infinito
    try:
        log_gamma_k = gammaln(k)
        numerador = (k - 1) * math.log(x) - (x / theta)
        denominador = log_gamma_k + k * math.log(theta)
        return math.exp(numerador - denominador)
    except (ValueError, OverflowError):  # ej. log(negativo) o exp(muy grande)
        return 0.0


def pmf_pascal(k_val, r_exitos, p_exito):  # k_val = número de fracasos
    if k_val < 0 or not isinstance(k_val, int):
        return 0
    if p_exito == 1:
        return 1.0 if k_val == 0 else 0.0
    if p_exito <= 0 or p_exito > 1:
        return 0.0  # p_exito debe estar en (0,1]
    try:
        # C(k+r-1, k) * p^r * (1-p)^k  o C(k+r-1, r-1)
        if k_val + r_exitos - 1 < r_exitos - 1:
            return 0  # Asegura que n >= k en C(n,k)
        coef_binomial = comb(
            k_val + r_exitos - 1, r_exitos - 1, exact=False
        )  # exact=False para float
        term_p = p_exito**r_exitos
        term_1_minus_p = (1 - p_exito) ** k_val
        return coef_binomial * term_p * term_1_minus_p
    except (ValueError, OverflowError):
        return 0.0


def pmf_binomial(k_val, n_ensayos, p_exito):
    if not (0 <= k_val <= n_ensayos and isinstance(k_val, int)):
        return 0
    if p_exito < 0 or p_exito > 1:
        return 0.0
    try:
        return (
            comb(n_ensayos, k_val, exact=False)
```

```
            * (p_exito**k_val)
            * ((1 - p_exito) ** (n_ensayos - k_val))
        )
    except (ValueError, OverflowError):
        return 0.0


def pmf_hipergeometrica(k_val, N_pop, K_ex_pop, n_muestra):
    if not isinstance(k_val, int):
        return 0
    min_k = max(0, n_muestra - (N_pop - K_ex_pop))
    max_k = min(n_muestra, K_ex_pop)
    if not (min_k <= k_val <= max_k):
        return 0.0
    try:
        term1 = comb(K_ex_pop, k_val, exact=False)
        term2 = comb(N_pop - K_ex_pop, n_muestra - k_val, exact=False)
        denominador = comb(N_pop, n_muestra, exact=False)
        if denominador == 0:
            return 0.0  # Evitar división por cero si C(N,n) es 0 (improbable con
    params válidos)
        return (term1 * term2) / denominador
    except (ValueError, OverflowError):
        return 0.0


def pmf_poisson(k_val, lam):
    if k_val < 0 or not isinstance(k_val, int):
        return 0
    if lam < 0:
        return 0.0
    if lam == 0:
        return 1.0 if k_val == 0 else 0.0
    try:
        # Usar logaritmos para estabilidad con k! grande
        log_pmf = k_val * math.log(lam) - lam - gammaln(k_val + 1)
        return math.exp(log_pmf)
    except (ValueError, OverflowError):
        return 0.0
```

rejection_method_generators.py

```
import math
from scipy.special import comb, gammaln  # comb para C(n,k), gammaln para log(
    Gamma(k))


# Funciones de Densidad/Probabilidad (FDP/FP) necesarias para el método de rechazo
def fdp_gamma(x, k, theta):  # Para Gamma
    if x < 0:
        return 0
    if x == 0:
        if k == 1:
            return 1.0 / theta if theta > 0 else float("inf")  # Exponencial
        if k > 1:
            return 0.0
        if k < 1:
            return float("inf")  # Tiende a infinito
    try:
        log_gamma_k = gammaln(k)
        numerador = (k - 1) * math.log(x) - (x / theta)
        denominador = log_gamma_k + k * math.log(theta)
        return math.exp(numerador - denominador)
    except (ValueError, OverflowError):  # ej. log(negativo) o exp(muy grande)
        return 0.0
```

```python


def pmf_pascal(k_val, r_exitos, p_exito):  # k_val = número de fracasos
    if k_val < 0 or not isinstance(k_val, int):
        return 0
    if p_exito == 1:
        return 1.0 if k_val == 0 else 0.0
    if p_exito <= 0 or p_exito > 1:
        return 0.0  # p_exito debe estar en (0,1]
    try:
        # C(k+r-1, k) * p^r * (1-p)^k  o C(k+r-1, r-1)
        if k_val + r_exitos - 1 < r_exitos - 1:
            return 0  # Asegura que n >= k en C(n,k)
        coef_binomial = comb(
            k_val + r_exitos - 1, r_exitos - 1, exact=False
        )  # exact=False para float
        term_p = p_exito**r_exitos
        term_1_minus_p = (1 - p_exito) ** k_val
        return coef_binomial * term_p * term_1_minus_p
    except (ValueError, OverflowError):
        return 0.0


def pmf_binomial(k_val, n_ensayos, p_exito):
    if not (0 <= k_val <= n_ensayos and isinstance(k_val, int)):
        return 0
    if p_exito < 0 or p_exito > 1:
        return 0.0
    try:
        return (
            comb(n_ensayos, k_val, exact=False)
            * (p_exito**k_val)
            * ((1 - p_exito) ** (n_ensayos - k_val))
        )
    except (ValueError, OverflowError):
        return 0.0


def pmf_hipergeometrica(k_val, N_pop, K_ex_pop, n_muestra):
    if not isinstance(k_val, int):
        return 0
    min_k = max(0, n_muestra - (N_pop - K_ex_pop))
    max_k = min(n_muestra, K_ex_pop)
    if not (min_k <= k_val <= max_k):
        return 0.0
    try:
        term1 = comb(K_ex_pop, k_val, exact=False)
        term2 = comb(N_pop - K_ex_pop, n_muestra - k_val, exact=False)
        denominador = comb(N_pop, n_muestra, exact=False)
        if denominador == 0:
            return 0.0  # Evitar división por cero si C(N,n) es 0 (improbable con
    params válidos)
        return (term1 * term2) / denominador
    except (ValueError, OverflowError):
        return 0.0


def pmf_poisson(k_val, lam):
    if k_val < 0 or not isinstance(k_val, int):
        return 0
    if lam < 0:
        return 0.0
    if lam == 0:
        return 1.0 if k_val == 0 else 0.0
    try:
```

```
        # Usar logaritmos para estabilidad con k! grande
        log_pmf = k_val * math.log(lam) - lam - gammaln(k_val + 1)
        return math.exp(log_pmf)
    except (ValueError, OverflowError):
        return 0.0
```