Examen – Modelación y Simulación

Bootstrapping, Transformada Inversa y Aceptación–Rechazo

Estudiante: Diederich Solis 22952

Repositorio: https://github.com/DiederichSolis/Examen_bootstrapping.git

a) Set de datos inicial

- Valores proporcionados: `[8, 8, 5, 9, 8, 10, 8, 7, 10, 1]`
- Media muestral: $\bar{x} = 7{,}400000$

b) Probabilidades normalizadas

- Vector de probabilidades: `[0.4, 0.17, 0.01, 0.04, 0.08, 0.01, 0.04, 0.02, 0.01, 0.21999999999999986`
- Suma verificada: $\sum p_i = 1{,}000000$

c) Explicación conceptual del Bootstrapping (100 palabras)

El bootstrapping es un método no paramétrico de remuestreo con reemplazo que permite estimar la distribución muestral de un estadístico (como la media) mediante:

- Generación de múltiples réplicas (B) de la muestra original
- Selección aleatoria con probabilidades dadas (via Transformada Inversa)
- Cálculo del estadístico en cada réplica
- Análisis de la distribución empírica resultante

Su ventaja principal es que no requiere supuestos distribucionales fuertes y proporciona estimaciones robustas de intervalos de confianza.

d) Implementación del Bootstrapping

d.i) Algoritmo de Transformada Inversa (discreta)

```python
import numpy as np

def inv_transform_discrete(cdf, u):
    """Transformada inversa para variables discretas.

    Args:
        cdf: Funci n de distribuci n acumulada
        u: Valor uniforme en [0,1]

    Returns:
         ndice  i que satisface CDF[i-1] < u     CDF[i]
    """
    return np.searchsorted(cdf, u, side='right')
```

Listing 1: Implementación Python

d.ii) Proceso de remuestreo

```
np.random.seed(42)  # Reproducibilidad
B = 10_000          # N mero de r plicas
N = len(data)       # Tama o muestral

# Centrado de datos
data_offset = data - np.mean(data)

# Generaci n de muestras bootstrap
bootstrap_means = np.empty(B)
U = np.random.rand(B * N)  # Vector de uniformes

pos = 0
for b in range(B):
    # Selecci n de  ndices  via Transformada Inversa
    idxs = [inv_transform_discrete(cdf, U[pos + i]) for i in range(N)]
    pos += N
    sample = data_offset[idxs]
    bootstrap_means[b] = np.mean(sample)
```

Listing 2: Bootstrapping de medias

d.iii) Resultados numéricos

- Desviación estándar de las medias: $\sigma_{\bar{x}} = 0{,}933237$
- Media de las medias bootstrap: $\hat{\mu} = -0{,}904390$ (esperado 0 por centrado)

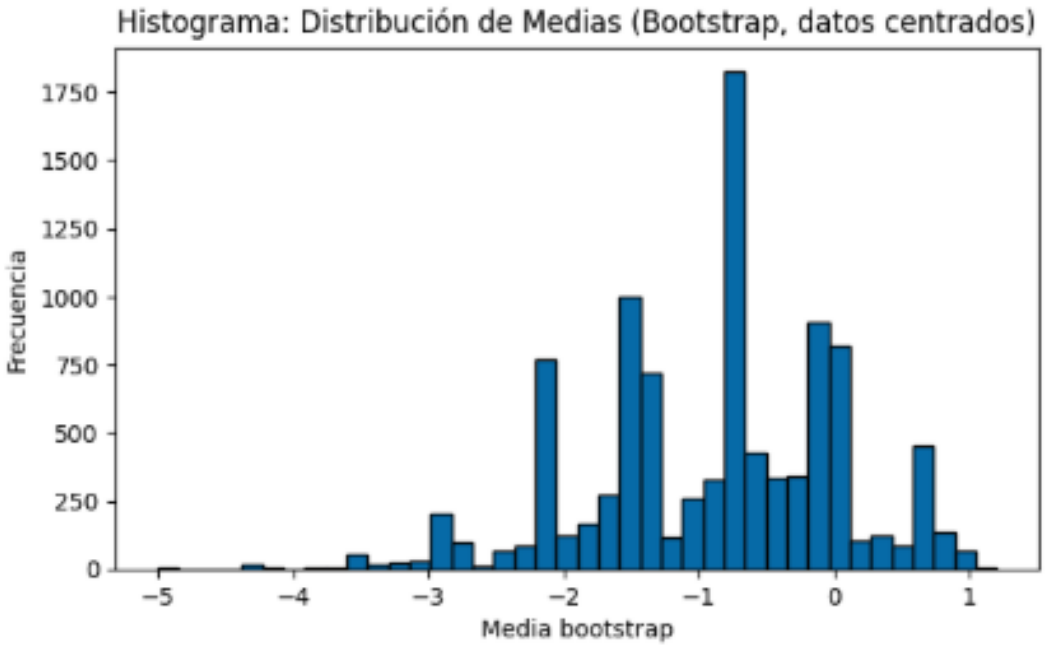

Figura 1: Distribución empírica de las medias bootstrap (datos centrados)

e) Probabilidades por rangos

Cuadro 1: Probabilidades empíricas por intervalo

Intervalo	Lím. Inferior	Lím. Superior	$\hat{P}$
1	-5.000	-3.760	0.0032
2	-3.760	-2.520	0.0443
3	-2.520	-1.280	0.3203
4	-1.280	-0.040	0.4528
5	-0.040	1.200	0.1794

f) Método de Aceptación-Rechazo

f.i) Especificación técnica

- Distribución objetivo: p_i (probabilidades por intervalo)
- Distribución propuesta: $q_i = \frac{1}{5}$ (uniforme discreta)
- Constante de mayoración: $c = \text{máx}\left(\frac{p_i}{q_i}\right) = 5 \times 0{,}4528 = 2{,}264$
- Criterio de aceptación: $u \leq \frac{p_j}{cq_j} = \frac{p_j}{0{,}4528}$

f.ii) Generador de números pseudoaleatorios

Implementación del LCG (Generador Congruencial Lineal):

$$X_{n+1} = (aX_n + c) \quad \text{mód } m$$

Parámetros:

$$a = 1103515245$$
$$c = 12345$$
$$m = 2^{31}$$
$$\text{Semilla} = 987654321$$

f.iii-iv) Resultados gráficos

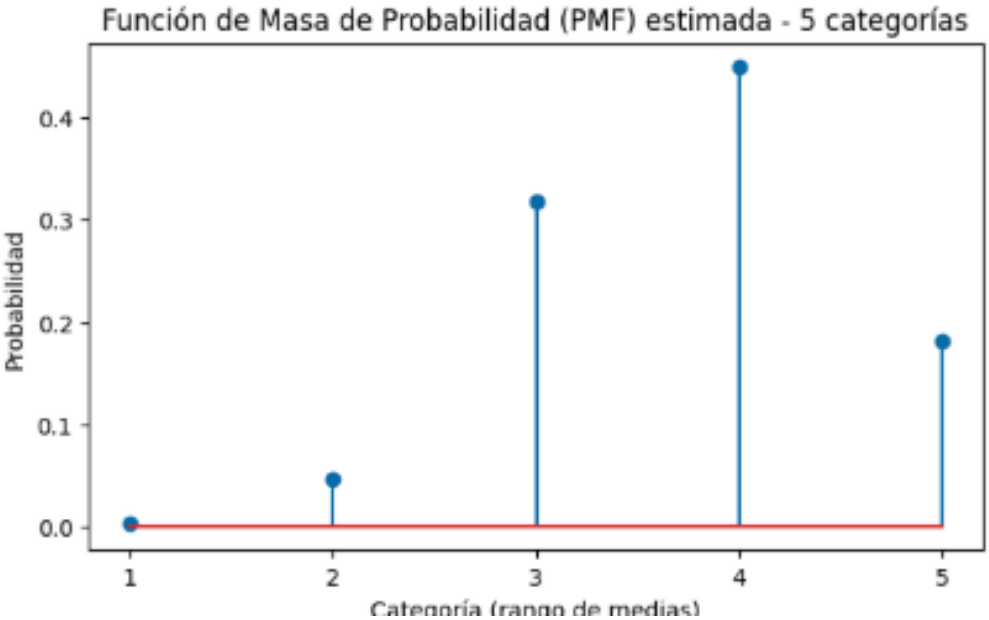

Figura 2: PMF estimada (10,000 muestras aceptadas)

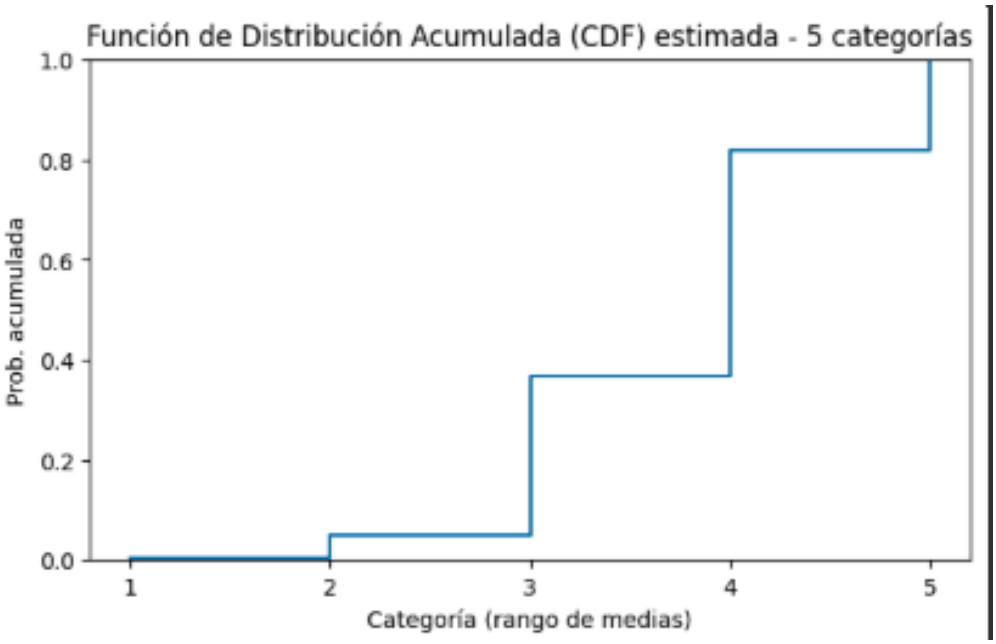

Figura 3: CDF empírica acumulada

g) Aplicación práctica en desarrollo de software

Esta metodología permite:

- **Pruebas A/B robustas**: Evaluación de cambios con intervalos de confianza no paramétricos
- **Simulación de usuarios**: Generación de comportamientos sintéticos para testing
- **Optimización de recursos**: Toma de decisiones basada en distribuciones empíricas
- **Implementación liviana**: Los algoritmos son computacionalmente eficientes para ejecución en dispositivos móviles

Anexo: Implementación completa de Aceptación-Rechazo

```
class LCG:
    """Generador congruencial lineal para n meros pseudoaleatorios"""
    def __init__(self, seed=987654321, a=1103515245, c=12345, m=2**31):
        self.state = seed % m
        self.params = {'a':a, 'c':c, 'm':m}

    def rand(self):
        """Genera entero en [0, m-1]"""
        a, c, m = self.params['a'], self.params['c'], self.params['m']
        self.state = (a * self.state + c) % m
        return self.state

    def rand_uniform(self):
        """Genera float en [0,1)"""
        return self.rand() / self.params['m']

```

```
def muestreo_ar(M, prob_bins, seed=987654321):
    """Genera M muestras via aceptaci n-rechazo"""
    lcg = LCG(seed=seed)
    maxp = prob_bins.max()
    accepted = []

    while len(accepted) < M:
        # Paso 1: Generar candidato de q (uniforme)
        j = int(lcg.rand_uniform() * 5)
        # Paso 2: Criterio de aceptaci n
        u = lcg.rand_uniform()
        if u <= prob_bins[j]/maxp:
            accepted.append(j)

    return np.array(accepted)
```

Listing 3: Algoritmo completo