ESMA 5015: Examen 2

Due on Abril 10, 2025

Damaris Santana

Alejandro Ouslan

Contents

1 Accept-Reject

Suponga que desea general variables aleatorias de una distribución $Gamma(\alpha, \beta)$ donde α no es necesariamente un entero. Decide usar el algoritmo **Accept-Reject** con la función candidata $Gamma(a, b)$.

1.1 Por que es necesario que $a < \alpha$ y $b > \beta$

Esto es para asegurar un buen candidato que se asurque lo mas posible a la función objetivo, esto es con el propósito de hacer el algoritmo mas eficiente.

1.2 Para $a = \lfloor \alpha \rfloor$, demuestre que M ocurre en $x = \frac{\alpha - \lfloor \alpha \rfloor}{\frac{1}{\beta} - \frac{1}{b}}$

$$
\begin{aligned}
x &= \sup \frac{f(x)}{g(x)} \\
&= \sup \frac{Gamma(\alpha, \beta)}{Gamma(a, b)} \\
&= \sup \frac{\frac{1}{\Gamma(\alpha)\beta^{\alpha}} x^{\alpha-1} e^{-\frac{x}{\beta}}}{\frac{1}{\Gamma(a)b^{a}} x^{a-1} e^{-\frac{x}{b}}} \\
&= \left(\frac{\frac{1}{\Gamma(\alpha)\beta^{\alpha}}}{\frac{1}{\Gamma(a)b^{a}}} \cdot \frac{x^{\alpha-1}}{x^{a-1}} \cdot \frac{e^{-\frac{x}{\beta}}}{e^{-\frac{x}{b}}} \right) \frac{d}{dx} \\
&= \left(x^{\alpha-a} \cdot e^{\frac{x}{b} - \frac{x}{\beta}} \right) \frac{d}{dx} \\
&= \left(x^{\alpha-1} e^{\frac{x}{b} - \frac{x}{\beta}} \right) \left(\left[\frac{1}{\beta} - \frac{1}{\beta} \right] + (\alpha - a)x^{-1} \right) = 0 \\
&= \left[\frac{1}{\beta} - \frac{1}{\beta} \right] + (\alpha - a)x^{-1} = 0 \\
&= \left[\frac{\alpha - a}{\frac{1}{\beta} - \frac{1}{b}} \right]_{a = \lfloor \alpha \rfloor} \\
&= \frac{\alpha - \lfloor \alpha \rfloor}{\frac{1}{\beta} - \frac{1}{b}}
\end{aligned}
$$

1.3 Para $a = \lfloor \alpha \rfloor$, encuentre el valor optimo de b

$$
\begin{aligned}
Mode_{target} &= Mode_{candidate} \\
(\alpha - 1)\beta &= (a - 1)b \\
b &= \frac{(\alpha - 1)\beta}{a - 1}
\end{aligned}
$$

Como $\alpha - 1 \approx \alpha$ y $a - 1 \approx a$

$$
b = \frac{\alpha\beta}{a}
$$

2 Implementación del algoritmo

2.1 Describa un algoritmo Accept-Reject para generar una variable aleatoria con distribución $Gamma(3/2, 1)$

$$\begin{aligned}
M &= \frac{f(x)}{f(x)} \\
&= \frac{f(x)}{g(x)} = \frac{\frac{2}{\sqrt{\pi}} x^{\frac{1}{2}} e^{-x}}{x e^{-x}} \\
&= \frac{2}{\sqrt{\pi}} \cdot \frac{1}{\sqrt{x}} \\
&= \frac{2}{\pi}
\end{aligned}$$

1. Generar un numero Y de la distribución $Gamma(2, 1)$.
2. Generar un numero uniforme $U \sim U(0, 1)$
3. Si $U \geq \frac{Y}{Mg(Y)}$
4. Repetir el proceso hasta aceptar un valor

2.2 Algoritmo en Python

Implementacion de python

```python
import numpy as np


# Función de densidad de la distribución objetivo (Gamma(3/2, 1))
def f(x):
    return (2 / np.sqrt(np.pi)) * np.sqrt(x) * np.exp(-x)


# Función de densidad de la distribución candidata (Gamma(1, 2))
def g(x):
    return x * np.exp(-x)


# Algoritmo de aceptación y rechazo
def accept_reject(n, seed=787):
    # Semilla para la generación de números aleatorios
    rng = np.random.default_rng(seed=seed)
    samples = []
    M = 2 / np.sqrt(np.pi)

    while len(samples) < n:
        # Paso 1: Generar una muestra de la distribución candidata (Gamma(1, 2))
        Y = rng.gamma(1, 2)

        # Paso 2: Generar una variable aleatoria uniforme U
        U = rng.uniform(0, 1)

        # Paso 3: Aceptar o rechazar
```

```
        if U <= f(Y) / (M * g(Y)):
            samples.append(Y)

    return np.array(samples)


if __name__ == "__main__":
    n_samples = 10000
    samples = accept_reject(n_samples)
```

2.3 Trafique el histograma de la distribución obtenida sobreponiendo la distribución deseada

Implementacion de python

```
from accept import *
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gamma


def main(n: int):
    samples = accept_reject(n)

    # Traficar el histograma
    x = np.linspace(0, 10, 1000)
    plt.hist(
        samples,
        bins=50,
        density=True,
        alpha=0.6,
        color="b",
        label="Histograma (muestras)",
    )
    plt.plot(x, gamma.pdf(x, 3/2, scale=1), 'r-', label='Distribución Gamma(3/2, 1)')

    plt.xlabel("x")
    plt.ylabel("Densidad")
    plt.legend()
    plt.title("Histograma de la Distribución Generada vs. Distribución Objetivo")
    plt.show()


if __name__ == "__main__":
    n_samples = 10000
    main(n_samples)
```

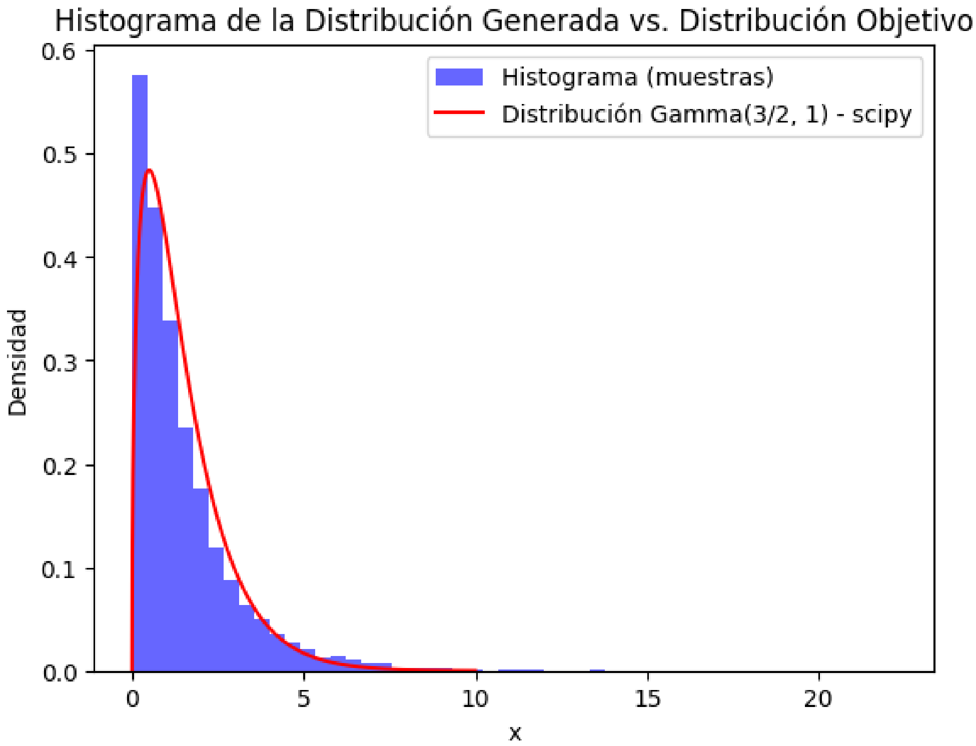

Figure 1: Histograma de la la gama simulada

2.4 Estime $E[X^2]$ y construya la gráfica de la convergencia de los running means.

$$
\begin{aligned}
E[X^2] &= \alpha(\alpha+1)\beta^2 \\
&= \frac{3}{2}(\frac{3}{2}1) = 3.75
\end{aligned}
$$

title

```
# Estimar E[X^2] y construir la gráfica de la convergencia de los "running means"
running_means = np.cumsum(samples) / np.arange(1, n_samples + 1) + (np.cumsum(samples) / np.arange(

# Graficar la convergencia
plt.plot(running_means, label="Running Mean")
plt.axhline(y=3.75, color="r", linestyle="--", label="Media Estimada")
plt.xlabel("Número de Muestras")
plt.ylabel("Running Mean")
plt.legend()
plt.title("Convergencia del Running Mean a la Media Estimada")
plt.show()
```

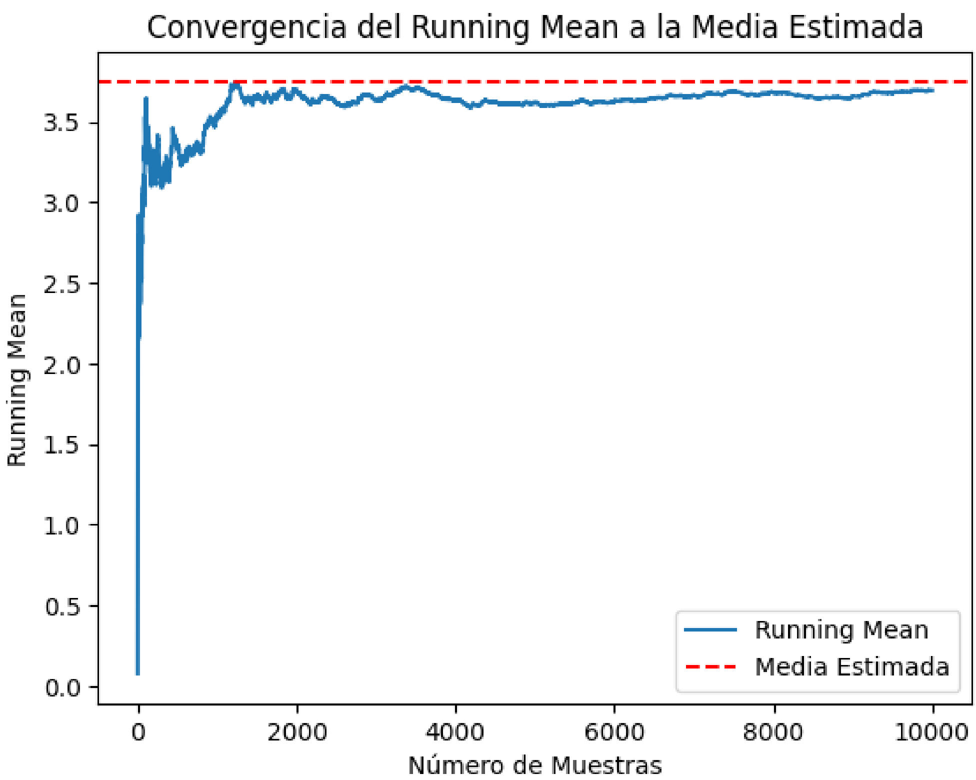

Figure 2: Histograma de la variable aleatoria

3 Importance Sampling

Usando Importance Sampling estime $E_f\left[\frac{X^5}{1+(X-3)^2}I[X \geq 0]\right]$, donde f es la distribución t con $v = 12$ Utilice las siguientes g:

3.1 Estimador importance Sampling

Para cada una de estas distribuciones presente el estimador que corresponde a la sumatoria definida por el método de **Importance Sampling** y que converge al valor esperado de interés

3.1.1 $Cauchy(0, 1)$

$$\begin{aligned} E_f\left[\frac{f(x)}{g(x)}p(x)\right] &= \frac{1}{N}\sum_{i=1}^{N}\frac{f(x)}{g(x)}p(x); \text{where } x \sim cauchy(0,1) \\ &= \frac{1}{N}\sum_{i=1}^{N}\frac{\frac{X^5}{1+(X-3)^2}I[X \geq 0]}{\frac{1}{\pi(1+x^2)}} \cdot student(12) \end{aligned}$$

3.1.2 $Normal(0, \frac{v}{v-2})$

$$E_f\left[\frac{f(x)}{g(x)}p(x)\right] = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x)}{g(x)}p(x); \text{where } x \sim normal(0, \frac{12}{12-2})$$
$$= \frac{1}{N}\sum_{i=1}^{N}\frac{\frac{X^5}{1+(X-3)^2}I[X \geq 0]}{}student(12)$$

3.1.3 $Exponencial(\lambda = 1)$

$$E_f\left[\frac{f(x)}{g(x)}p(x)\right] = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x)}{g(x)}p(x); \text{where } x \sim exponential(1)$$
$$= \frac{1}{N}\sum_{i=1}^{N}\frac{\frac{X^5}{1+(X-3)^2}I[X \geq 0]}{g(x)}student(12)$$

3.2 Estimador Monte Carlo

Para cada uno presente el estimador que corresponde a la sumatoria definida por el método de Integración Monte Carlos y que converge al valor esperado de interés.

$$MC = \frac{1}{N}\sum_{i=1}^{N} f(x); \text{where } x \sim student(12)$$
$$= \frac{1}{N}\sum_{i=1}^{N} f\frac{X^5}{1+(X-3)^2}I[X \geq 0]$$

3.3 Implementación

3.3.1 Código para Monte Carlos

Implementacion de Monte Carlos

```
import numpy as np
import scipy.stats as stats


def h(X):
    return (X**5) / (1 + (X - 3) ** 2) * (X >= 0)


if __name__ == "__main__":
    samples = stats.t.rvs(df=12, size=1000)
    monte_carlo_estimate = np.mean(h(samples))
```

3.3.2 Código para $Cauchy(0, 1)$

Implementacion de distribución cauchy

```
import numpy as np
import scipy.stats as stats
```

```python
def h(X):
    return (X**5) / (1 + (X - 3) ** 2) * (X >= 0)


def g_cauchy(x):
    return 1 / (np.pi * (1 + x**2))


def importance_sampling_cauchy(samples, v):
    f_samples = stats.t.pdf(samples, df=v)
    g_samples = g_cauchy(samples)
    weights = f_samples / g_samples
    h_values = h(samples)

    estimate = np.mean(weights * h_values)

    return estimate


if __name__ == "__main__":
    samples_cauchy = np.random.standard_cauchy(size=1000)
    cauchy_estimate = importance_sampling_cauchy(samples_cauchy, 12)
    print(cauchy_estimate)
```

3.3.3 Código para$Normal(0, \frac{v}{v-2})$

Implementacion de distribución normal

```python
import numpy as np
import scipy.stats as stats


def h(X):
    return (X**5) / (1 + (X - 3) ** 2) * (X >= 0)


def g_normal(x, v):
    return stats.norm.pdf(x, loc=0, scale=np.sqrt(v / (v - 2)))


def importance_sampling_normal(samples, v):
    f_samples = stats.t.pdf(samples, df=v)
    g_samples = g_normal(samples, v)

    weights = f_samples / g_samples
    h_values = h(samples)
    return np.mean(weights * h_values)


if __name__ == "__main__":
```

```
    samples_normal = np.random.normal(0, np.sqrt(12 / (12 - 2)), size=100)

    normal_estimate = importance_sampling_normal(samples_normal, 12)
    print(normal_estimate)
```

3.3.4 Código para *Exponencial*$(\lambda = 1)$

Implementacion de distribución Exponencial

```
import numpy as np
import scipy.stats as stats


def h(X):
    return (X**5) / (1 + (X - 3) ** 2) * (X >= 0)


def g_exponential(x):
    return np.exp(-x) * (x >= 0)


def importance_sampling_exponential(samples):
    f_samples = stats.t.pdf(samples, df=12)

    g_samples = g_exponential(samples)

    weights = f_samples / g_samples
    h_values = h(samples)

    estimate = np.mean(weights * h_values)

    return estimate


if __name__ == "__main__":
    samples_exponential = np.random.exponential(1, size=1000)
    exponential_estimate = importance_sampling_exponential(samples_exponential)
    print(exponential_estimate)
```

3.4 Gráficas

Construya un asola gracia y presente la convergencia de los running menas para los cuatro estimadores. Compare la varianza empírica de los cuatro estimadores

Implementacion de distribución Exponencial

```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

normal_samples = []
```

```python
cauchy_samples = []
exp_samples = []
mc_samples = []

for i in range(1000):
    samples_cauchy = np.random.standard_cauchy(size=1000)
    cauchy_samples.append(importance_sampling_cauchy(samples_cauchy, 12))

    samples_exponential = np.random.exponential(1, size=1000)
    exp_samples.append(importance_sampling_exponential(samples_exponential))

    samples = stats.t.rvs(df=12, size=1000)
    mc_samples.append(np.mean(h(samples)))

    samples_normal = np.random.normal(0, np.sqrt(12 / (12 - 2)), size=100)

    normal_samples.append(importance_sampling_normal(samples_normal, 12))


normal_data = np.array(normal_samples)
cauchy_data = np.array(cauchy_samples)
mc_data = np.array(mc_samples)
exp_data = np.array(exp_samples)

# Estimar E[X^2] y construir la gráfica de la convergencia de los "running means"
running_normal = np.cumsum(normal_data) / np.arange(1, 1000 + 1)
running_cauchy = np.cumsum(cauchy_data) / np.arange(1, 1000 + 1)
running_exp = np.cumsum(exp_data) / np.arange(1, 1000 + 1)
running_mc = np.cumsum(mc_data) / np.arange(1, 1000 + 1)

# Graficar la convergencia
plt.plot(running_normal, label="Running normal")
plt.plot(running_cauchy, label="Running cauchy")
plt.plot(running_exp, label="Running exp")
plt.plot(running_mc, label="Running mc")

plt.axhline(y=4.64, color="r", linestyle="--", label="Media Real")
plt.xlabel("Número de Muestras")
plt.ylabel("Running Mean")
plt.legend()
plt.title("Convergencia del Running Mean a la Media Estimada")
plt.savefig("running_all.png")
```

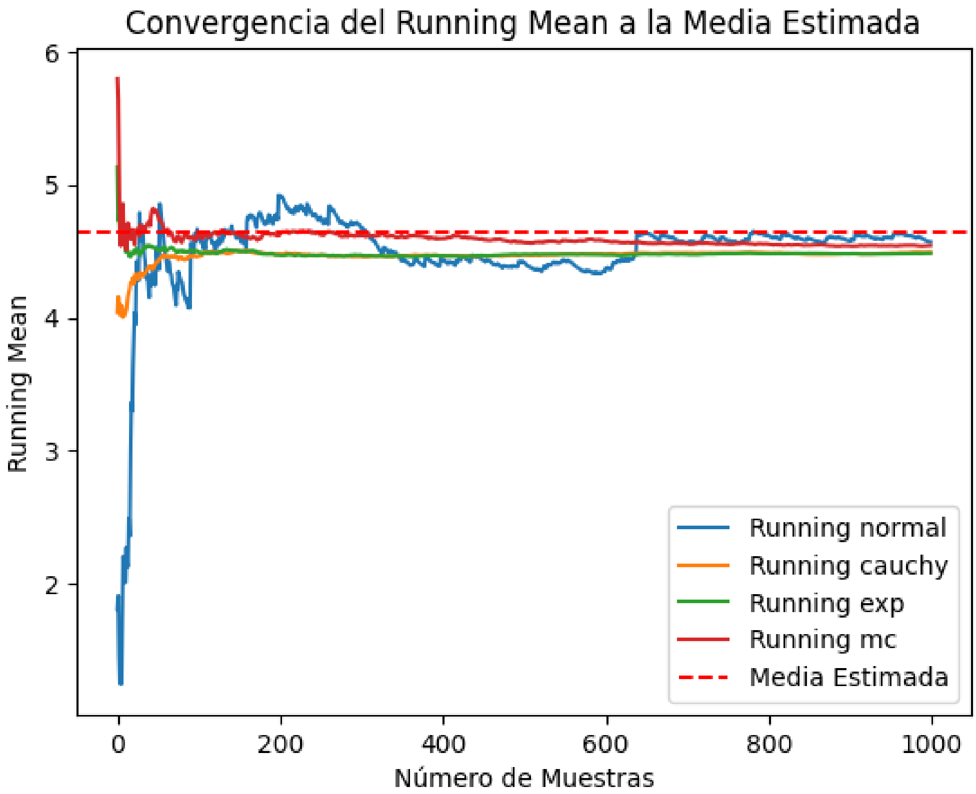

Figure 3: Histograma de la variable aleatoria