Actividad 3.2

Curso: Temas Selectos I: O25 LAT4032 1

Profesor: Rubén Blancas Rivera

Mayren Herrera Vargas, ID: 173802
Sofia Graham Coello, ID: 174291
Heriberto Espino Montelongo, ID: 175199

Universidad de las Américas Puebla

25 de Octubre de 2025

Índice

Importación de Librerías

[1]:
```
import numpy as np
from math import erf, sqrt, comb, floor
```

Ejercicio 1

Sea $\phi(x)$ la densidad de $\mathcal{N}(0,1)$. Se sabe que su cuantil del 95 % satisface

$$\int_{-\infty}^{c_p} \phi(x)\,dx = 0.95, \qquad c_p \approx 1.65.$$

Escriba un programa que utilice el *método clásico de Monte Carlo* para corroborar este resultado, realizando la integración numérica en el intervalo acotado $(-4,\ 1.65)$.

Planteamiento

Queremos aproximar

$$\theta = \int_{-4}^{1.65} \phi(x)\,dx \approx \int_{-\infty}^{1.65} \phi(x)\,dx,$$

pues $\Phi(-4)$ es despreciable. Para aplicar Monte Carlo clásico (hit-or-miss) se elige un rectángulo

$$R = [a,b] \times [0,c], \quad a = -4,\ b = 1.65,\ c = \max_{x\in[a,b]} \phi(x) = \phi(0) = \frac{1}{\sqrt{2\pi}}.$$

Sea $(X,Y) \sim \text{Unif}(a,b) \times \text{Unif}(0,c)$; entonces

$$p = \mathbb{P}\{Y \leqslant \phi(X)\} = \frac{1}{c(b-a)} \int_a^b \phi(x)\,dx = \frac{\theta}{c(b-a)}.$$

Con N puntos bajo la curva en una muestra de tamaño n, el estimador clásico es

$$\hat{\theta} = c(b-a)\,\hat{p} = c(b-a)\,\frac{N}{n},$$

insesgado y con varianza $\text{Var}(\hat{\theta}) = \theta\,[c(b-a) - \theta]/n$.

Pseudocódigo (Monte Carlo clásico)

1. **Entradas:** $a = -4, b = 1.65, c = \phi(0)$, tamaño n grande.
2. **Para** $i = 1, \dots, n$:
   a) Generar $X_i \sim \text{Unif}(a,b)$ y $Y_i \sim \text{Unif}(0,c)$ de forma independiente.
   b) Definir $I_i = \mathbf{1}\{Y_i \leqslant \phi(X_i)\}$.
3. Calcular $N = \sum_{i=1}^{n} I_i$, $\hat{p} = N/n$ y
$$\hat{\theta} = c\,(b-a)\,\hat{p}.$$
4. Reportar $\hat{\theta}$ y un IC al 95 %: $\hat{\theta} \pm 1.96\,\widehat{\text{SE}}$.

Código

[2]:
```
def phi(x):
    """Densidad normal estándar  (x)."""
    return (1/np.sqrt(2*np.pi)) * np.exp(-x**2/2)

# Parámetros del rectángulo
```

```
a, b = -4.0, 1.65                       # intervalo acotado
c = 1/np.sqrt(2*np.pi)                  # c = max  (x) =  (0)
n = 300_000                             # número de puntos
rng = np.random.default_rng(7)          # semilla para reproducibilidad

# Muestreo uniforme en el rectángulo [a,b]×[0,c]
x = rng.uniform(a, b, n)
y = rng.uniform(0.0, c, n)

# Conteo de puntos bajo la curva
n0 = np.sum(y <= phi(x))

# Estimador clásico:  ^= c (b-a) · (n0/n)
rect_area = c * (b - a)
p_hat = n0 / n
theta_hat = rect_area * p_hat

# Error estándar plug-in usando Var( ) =  /n [ c(b-a) -   ] (desconocido   sustituimos  )
var_hat = (theta_hat / n) * (rect_area - theta_hat)
se_hat = np.sqrt(var_hat)

# Valor "teórico" con CDF normal vía erf (para comparar)
def Phi(z):  # CDF normal estándar
    return 0.5*(1 + erf(z/sqrt(2)))

true_full = Phi(1.65)           # ∫_{-∞}^{1.65}
left_tail = Phi(-4.0)           # ∫_{-∞}^{-4}   (muy pequeño)
true_trunc = true_full - left_tail   # ∫_{-4}^{1.65}

# IC aproximado al 95%
ci_low = theta_hat - 1.96*se_hat
ci_high = theta_hat + 1.96*se_hat

print(f"Estimación Monte Carlo (truncada [-4,1.65]): {theta_hat:.6f}")
print(f"Error estándar (plug-in): {se_hat:.6f}")
print(f"IC 95% ( ^± 1.96·SE): [{ci_low:.6f}, {ci_high:.6f}]")
print(f"Valor teórico ∫_(-∞)^(1.65)   = {true_full:.6f}")
print(f"Cola izquierda  (-4) = {left_tail:.8f}     valor truncado ≈ {true_trunc:.6f}")
```

```
Estimación Monte Carlo (truncada [-4,1.65]): 0.953715
Error estándar (plug-in): 0.002033
IC 95% ( ^± 1.96·SE): [0.949730, 0.957700]
Valor teórico ∫_(-∞)^(1.65)   = 0.950529
Cola izquierda  (-4) = 0.00003167     valor truncado ≈ 0.950497
```

Ejercicio 2

Escriba el algoritmo para llevar a cabo la integración Monte Carlo en su procedimiento clásico cuando la función a integrar es:

(a) $g(x) = c$ (constante), para $a < x < b$.

(b) $g(x) = 2x$ para $0 < x < 1$, cuando se toma $c = 1$.

Planteamiento

Sea $g : [a, b]$ E $\mathbb{R}_+$ acotada por arriba, es decir, existe $c > 0$ tal que $0 \leqslant g(x) \leqslant c$ para todo $x \in (a, b)$. Considere una variable aleatoria (X, Y) con distribución uniforme independiente en el rectángulo

$$R = [a, b] \times [0, c], \qquad X \sim \text{Unif}(a, b), \;\; Y \sim \text{Unif}(0, c).$$

Entonces, por geometría,

$$p \;=\; \Pr\{Y \leqslant g(X)\} = \frac{1}{c(b-a)} \int_a^b g(x)\, dx = \frac{\theta}{c(b-a)}, \qquad \theta = \int_a^b g(x)\, dx.$$

Si tomamos una muestra $\{(X_i, Y_i)\}_{i=1}^n$ i.i.d. de (X, Y) y definimos $N = \sum_{i=1}^n \mathbf{1}\{Y_i \leqslant g(X_i)\} \sim \text{Bin}(n, p)$, el estimador clásico es

$$\hat{\theta} = c(b-a)\,\hat{p} = c(b-a)\,\frac{N}{n}.$$

Es insesgado y su varianza es

$$\text{Var}(\hat{\theta}) = \frac{\theta}{n}\,(c(b-a) - \theta).$$

En la práctica, como θ es desconocido, se sustituye por $\hat{\theta}$ para obtener un error estándar tipo plug-in:

$$\widehat{\text{SE}}(\hat{\theta}) = \sqrt{\frac{\hat{\theta}}{n}\,(c(b-a) - \hat{\theta})}.$$

Regla crucial: el método clásico requiere una cota válida c con $0 \leqslant g(x) \leqslant c$ en todo (a, b). Elegir c lo más pequeño posible (respetando $g \leqslant c$) reduce la varianza.

Inciso (a): $g(x) = c$ constante en (a, b)

Elección del rectángulo. Como $g(x) \equiv c$, la cota mínima válida es c mismo. Tomamos $R = [a, b] \times [0, c]$.

Probabilidad y estimación. Para todo i, $Y_i \leqslant g(X_i)$ ocurre con probabilidad 1 porque $Y_i \in [0, c]$ y $g(X_i) = c$. Entonces $N = n$ y

$$\hat{\theta} = c(b-a)\,\frac{N}{n} = c(b-a),$$

que coincide exactamente con el valor verdadero $\theta = \displaystyle\int_a^b c\, dx = c(b-a)$. En consecuencia, $\text{Var}(\hat{\theta}) = 0$.

Algoritmo (a).

1. Entradas: $a < b$, constante $c \geqslant 0$, tamaño n.
2. Para $i = 1, \ldots, n$: generar $X_i \sim \text{Unif}(a, b)$, $Y_i \sim \text{Unif}(0, c)$; $I_i = \mathbf{1}\{Y_i \leqslant c\}$ (siempre 1).
3. $N = \sum I_i = n$ y $\hat{\theta} = c(b-a)$.

Inciso (b): $g(x) = 2x$ en $(0, 1)$

Aquí $\max_{x\in(0,1)} g(x) = 2$, de modo que la *cota mínima válida* es $c = 2$. Con $c = 2$:

$$\theta = \int_0^1 2x\,dx = 1, \quad c(b-a) = 2 \cdot 1 = 2, \quad p = \frac{\theta}{c(b-a)} = \frac{1}{2}.$$

El estimador es $\hat{\theta} = 2\,\frac{N}{n}$ con $N \sim \text{Bin}(n, \frac{1}{2})$, y $\text{Var}(\hat{\theta}) = \frac{1}{n}(2-1) = \frac{1}{n}$.

Qué sucede si "se toma $c = 1$". La condición $g \leqslant c$ se viola porque $g(x) = 2x > 1$ cuando $x > 0.5$. El rectángulo $[0, 1] \times [0, 1]$ *recorta* la región bajo la curva y lo que se estima es

$$\tilde{\theta} = \int_0^1 \text{mín}\{2x, 1\}\,dx = \int_0^{1/2} 2x\,dx + \int_{1/2}^1 1\,dx = \frac{1}{4} + \frac{1}{2} = \frac{3}{4}\ \text{q}\ \theta,$$

es decir, el procedimiento resulta sesgado. Por lo tanto, para el método clásico **debe** usarse $c \geqslant 2$.

Algoritmo (b) correcto, con $c = 2$.

1. Entradas: $a = 0, b = 1, c = 2$, tamaño n.
2. Para $i = 1, \ldots, n$: generar $X_i \sim \text{Unif}(0, 1)$, $Y_i \sim \text{Unif}(0, 2)$; $I_i = \mathbf{1}\{Y_i \leqslant 2X_i\}$.
3. $N = \sum I_i$, $\hat{\theta} = 2 \cdot (N/n)$; opcionalmente, estimar $\widehat{\text{SE}}$ y un IC al 95 % vía $\hat{\theta} \pm 1.96\,\widehat{\text{SE}}$.

Pseudocódigo general

1. **Entradas:** intervalo (a, b); cota $c > 0$ con $0 \leqslant g(x) \leqslant c$ en (a, b); tamaño muestral n.
2. **Para** $i = 1, \ldots, n$**:**
   a) Generar $X_i \sim \text{Unif}(a, b)$ y $Y_i \sim \text{Unif}(0, c)$, independientes.
   b) Definir $I_i = \mathbf{1}\{Y_i \leqslant g(X_i)\}$.
3. **Conteo:** $N = \sum_{i=1}^n I_i$ y $\hat{p} = N/n$.
4. **Estimación:** $\hat{\theta} = c\,(b-a)\,\hat{p}$.
5. **Error (opcional):** $\widehat{\text{SE}} = \sqrt{\frac{\hat{\theta}}{n}(c(b-a) - \hat{\theta})}$; **IC 95 %:** $\hat{\theta} \pm 1.96\,\widehat{\text{SE}}$.

Código

[3]:
```python
def mc_clasico_hit_or_miss(g, a, b, c, n=200_000, seed=123):
    """
    Método Monte Carlo clásico (hit-or-miss) para ∫_a^b g(x) dx con 0 <= g(x) <= c.
    Devuelve (theta_hat, se_hat, n0, rect_area).
    """
    rng = np.random.default_rng(seed)
    x = rng.uniform(a, b, size=n)
    y = rng.uniform(0.0, c, size=n)
    under = y <= g(x)
    n0 = int(np.sum(under))
    rect_area = c * (b - a)
    p_hat = n0 / n
    theta_hat = rect_area * p_hat
    # Var( ) ≈  7n * (c(b-a) -  )  (plug-in)
    var_hat = (theta_hat / n) * (rect_area - theta_hat)
    se_hat = np.sqrt(max(var_hat, 0.0))
    return theta_hat, se_hat, n0, rect_area

# (a) g(x)=c0 constante
a, b, c0 = 0.0, 3.0, 5.0
```

```
g_const = lambda x: c0*np.ones_like(x)
theta_hat_a, se_hat_a, _, _ = mc_clasico_hit_or_miss(g_const, a, b, c=c0, n=100_000, seed=7)
print("(a) g(x)=c0 en (a,b):    ^=", theta_hat_a, "  SE ≈", se_hat_a, "  (valor real =", c0*(b-a), ")")

# (b) g(x)=2x en (0,1) — caso correcto: c=2
g_lin = lambda x: 2.0*x
theta_hat_b_ok, se_hat_b_ok, _, _ = mc_clasico_hit_or_miss(g_lin, 0.0, 1.0, c=2.0, n=200_000, seed=7)
print("(b) g(x)=2x, c=2:        ^≈", round(theta_hat_b_ok,6), "  SE ≈", round(se_hat_b_ok,6), "  (valor real = 1)")

# (b) Si alguien usa c=1 (no válido): muestra el sesgo hacia 3/4
theta_hat_b_bad, se_hat_b_bad, _, _ = mc_clasico_hit_or_miss(g_lin, 0.0, 1.0, c=1.0, n=200_000, seed=7)
print("(b) g(x)=2x, c=1 (!)     ^≈", round(theta_hat_b_bad,6), "  SE ≈", round(se_hat_b_bad,6), "  (valor 'truncado' = 0.
↪75)")
```

```
(a) g(x)=c0 en (a,b):    ^= 15.0   SE ≈ 0.0   (valor real = 15.0 )
(b) g(x)=2x, c=2:        ^≈ 1.00249   SE ≈ 0.002236   (valor real = 1)
(b) g(x)=2x, c=1 (!)     ^≈ 0.75134   SE ≈ 0.000967   (valor 'truncado' = 0.75)
```

Ejercicio 3

Elabore un programa de cómputo que utilice el método clásico de integración Monte Carlo para calcular el volumen de una esfera de radio 1. Para corroborar su resultado, recuerde que el volumen de una esfera de radio r es $\frac{4}{3}\pi r^3$.

Planteamineto. Sea $B = \{(x, y, z) \in \mathbb{R}^3 : x^2+y^2+z^2 \leqslant r^2\}$ la esfera de radio r y encajémosla en el cubo $C = [-r, r]^3$ de volumen $V_C = (2r)^3$. Si tomamos $(X, Y, Z) \sim \text{Unif}([-r, r]^3)$ y definimos $N = \#\{(X_i, Y_i, Z_i) \in B\}$ sobre una muestra i.i.d. de tamaño n, entonces

$$p = \Pr\{(X, Y, Z) \in B\} = \frac{\text{Vol}(B)}{\text{Vol}(C)} = \frac{V}{V_C}, \quad \text{de modo que} \quad V = V_C\, p.$$

El estimador clásico *hit-or-miss* resulta de reemplazar p por $\hat{p} = N/n$:

$$\widehat{V} \;=\; V_C\,\hat{p} \;=\; V_C\,\frac{N}{n}.$$

Esta construcción es la extensión multidimensional del método clásico: en general, si $0 \leqslant g(x, y) \leqslant c$ en un rectángulo $(a_1, b_1) \times (a_2, b_2)$, se tiene $p = \frac{1}{c(b_1-a_1)(b_2-a_2)} \iint g$ y $\theta = c(b_1 - a_1)(b_2 - a_2)p$; por analogía, en 3D el volumen buscado es el volumen del contenedor por la fracción de puntos "favorables".

Como $N \sim \text{Bin}(n, p)$, la varianza del estimador es

$$\text{Var}(\widehat{V}) \;=\; V_C^2\,\frac{p(1-p)}{n} \quad \text{(equiv. plug-in)} \quad \widehat{\text{Var}}(\widehat{V}) \;=\; \frac{\widehat{V}}{n}\,(V_C - \widehat{V}),$$

y un IC aproximado al 95 % es $\widehat{V} \pm 1.96\,\widehat{\text{SE}}$, con $\widehat{\text{SE}} = \sqrt{\widehat{\text{Var}}(\widehat{V})}$. Estas propiedades provienen directamente del caso 1D del método clásico (insesgado, varianza $\theta\,[c(b-a) - \theta]/n$) y se trasladan a la versión geométrica en 3D.

Para $r = 1$ se usa $C = [-1, 1]^3$ con $V_C = 8$ y se compara con el valor exacto $V_{\text{exacto}} = \frac{4}{3}\pi r^3$.

Código

[4]:
```python
def volumen_esfera_mc(r=1.0, n=1_000_000, seed=42):
    """
    Volumen de una esfera de radio r mediante Monte Carlo clásico (hit-or-miss) en 3D.
    Idea: muestrear uniforme en el cubo [-r, r]^3 y contar la fracción de puntos
    que cae dentro de la esfera x^2 + y^2 + z^2 <= r^2.
    """
    rng = np.random.default_rng(seed)

    # 1) Volumen del cubo contenedor: V_cubo = (2r)^3
    a, b = -r, r
    V_cubo = (b - a) ** 3

    # 2) Muestreo uniforme en el cubo
    x = rng.uniform(a, b, size=n)
    y = rng.uniform(a, b, size=n)
    z = rng.uniform(a, b, size=n)

    # 3) Indicador: ¿cayó dentro de la esfera?
    inside = (x*x + y*y + z*z) <= (r*r)
    n0 = int(np.sum(inside))

    # 4) Estimador clásico: V̂ = V_cubo * (n0/n)
    p_hat = n0 / n
```

```
    V_hat = V_cubo * p_hat

    # 5) Error estándar (plug-in) por binomial: Var(V̂) = V_cubo^2 * p(1-p)/n
    #    Forma equivalente: Var(V̂) = (V_hat/n) * (V_cubo - V_hat)
    var_hat = (V_hat / n) * (V_cubo - V_hat)
    se_hat = np.sqrt(max(var_hat, 0.0))

    # 6) Valor exacto y un IC 95% (approx)
    V_true = (4.0/3.0) * np.pi * (r**3)
    ci = (V_hat - 1.96*se_hat, V_hat + 1.96*se_hat)

    return {
        "V_hat": V_hat,
        "SE": se_hat,
        "n": n,
        "r": r,
        "V_cubo": V_cubo,
        "n0": n0,
        "p_hat": p_hat,
        "IC95": ci,
        "V_true": V_true,
    }

# Ejemplo de uso:
res = volumen_esfera_mc(r=1.0, n=1_000_000, seed=7)
print(f"Volumen estimado = {res['V_hat']:.5f}")
print(f"Error estándar   = {res['SE']:.6f}")
print(f"IC 95%           = [{res['IC95'][0]:.5f}, {res['IC95'][1]:.5f}]")
print(f"Volumen exacto   = {res['V_true']:.5f}")
```

```
Volumen estimado = 4.19170
Error estándar   = 0.003995
IC 95%           = [4.18387, 4.19953]
Volumen exacto   = 4.18879
```

Ejercicio 4

Usando simulación, aproxime el área de la región

$$\left\{(x,y) : -1 < x < 1,\ y > 0,\ \sqrt{1-2x^2} < y < \sqrt{1-2x^4}\right\}.$$

Planteamiento. Sea $R = [a,b] \times [0,1]$ un rectángulo que contenga a la región pedida; por ejemplo podemos reducir la varianza tomando $a = -1/\sqrt{2}, b = 1/\sqrt{2}$ (pues fuera de ese rango $1-2x^2 < 0$ y la región es vacía). Si $(X,Y) \sim \text{Unif}(R)$ y $I = \mathbf{1}\{\sqrt{1-2X^2} < Y < \sqrt{1-2X^4}\}$, entonces

$$p = \Pr(I=1) = \frac{\text{Área(región)}}{\text{Área}(R)} \qquad \Longrightarrow \qquad \widehat{A} = \text{Área}(R) \cdot \frac{1}{n}\sum_{i=1}^{n} I_i.$$

Este es el método clásico *hit-or-miss*: muestrear puntos uniformes en R y contar cuántos caen dentro de la región; es insesgado y $\text{Var}(\widehat{A}) = \widehat{A}\,(\text{Área}(R) - \widehat{A})/n$ (plug-in). También puede usarse la media muestral en $[a,b]$ con $h(x) = \sqrt{1-2x^4} - \sqrt{1-2x^2}$ (no negativa en $[a,b]$): $\widehat{A}_{\text{mm}} = (b-a)\frac{1}{n}\sum_{i=1}^{n} h(X_i)$, que típicamente tiene menor varianza.

Código

[5]:
```python
def area_region_hit_or_miss(n=300_000, seed=123, tight=True):
    """
    Estima el área de {(x,y): -1<x<1, y>0, sqrt(1-2x^2) < y < sqrt(1-2x^4)}
    por 'hit-or-miss'.

    Si tight=True usa el rectángulo [-1/sqrt(2), 1/sqrt(2)] x [0, 1],
    que contiene a la región y reduce varianza.
    """
    rng = np.random.default_rng(seed)
    if tight:
        a, b = -1/np.sqrt(2), 1/np.sqrt(2)
    else:
        a, b = -1.0, 1.0
    c, d = 0.0, 1.0                      # en y basta [0,1], pues y<sqrt(1-2x^4)≤1
    area_rect = (b - a) * (d - c)

    # Muestras uniformes en el rectángulo
    x = rng.uniform(a, b, size=n)
    y = rng.uniform(c, d, size=n)

    # Radicandos (para evitar sqrt de negativos)
    r1 = 1 - 2*x**2
    r2 = 1 - 2*x**4
    valid = (r1 >= 0) & (r2 >= 0)

    y_low  = np.zeros_like(x)
    y_high = np.zeros_like(x)
    y_low[valid]  = np.sqrt(r1[valid])
    y_high[valid] = np.sqrt(r2[valid])

    inside = valid & (y > y_low) & (y < y_high)
    p_hat = np.mean(inside)
    area_hat = area_rect * p_hat

    # SE (plug-in) para hit-or-miss: Var = A/n * (AreaRect - A)
    var_hat = (area_hat / n) * (area_rect - area_hat)
    se_hat = np.sqrt(max(var_hat, 0.0))
    ci95 = (area_hat - 1.96*se_hat, area_hat + 1.96*se_hat)
```

```
    return {"area_hat": area_hat, "SE": se_hat, "IC95": ci95,
            "n": n, "rect_area": area_rect, "p_hat": p_hat}

# Ejemplo rápido:
res = area_region_hit_or_miss(n=400_000, seed=7, tight=True)
res
```

[5]:
```
{'area_hat': np.float64(0.2264580177428037),
 'SE': np.float64(0.0008200255578334128),
 'IC95': (np.float64(0.2248507676494502), np.float64(0.2280652678361572)),
 'n': 400000,
 'rect_area': np.float64(1.414213562373095),
 'p_hat': np.float64(0.16013)}
```

Ejercicio 5

Suponga que se tienen 3 tipos de riesgos en una aseguradora. Cada riesgo puede provocar una reclamación cuya cantidad en miles de pesos se modela con una variable aleatoria X_i, para $i = 1, 2, 3$. Los tres riesgos se consideran independientes y, en cada caso, $X_i \sim \text{Unif}(0, 15)$. La aseguradora desea calcular el valor x tal que

$$\Pr(X_1 + X_2 + X_3 \leqslant x) \;=\; 0.95.$$

Exprese la probabilidad anterior como la integral triple que corresponde. Aproxime el valor de x realizando simulaciones de X_1, X_2, X_3 y probando distintos valores de x.

Planteamiento

Sean $X_1, X_2, X_3 \overset{\text{i.i.d.}}{\sim} \text{Unif}(0, 15)$ y $S = X_1 + X_2 + X_3$. Para $x \in \mathbb{R}$, la probabilidad requerida puede escribirse como volumen bajo el indicador en el cubo $[0, 15]^3$:

$$\Pr(S \leqslant x) = \frac{1}{15^3} \iiint_{[0,15]^3} \mathbf{1}\{u + v + w \leqslant x\}\, du\, dv\, dw,$$

es decir, la *integral triple* de la función $g(u, v, w) = \mathbf{1}\{u + v + w \leqslant x\}$ en el dominio acotado con densidad uniforme (caso multidimensional del método clásico). *Equivalencias Monte Carlo:* $\Pr(S \leqslant x) = E[\mathbf{1}\{S \leqslant x\}]$, de modo que un estimador insesgado es la media muestral de los indicadores.

Algoritmo de simulación

1. Genere una muestra i.i.d. $X_{i1}, X_{i2}, X_{i3} \sim \text{Unif}(0, 15)$ para $i = 1, \ldots, n$ y compute $S_i = X_{i1} + X_{i2} + X_{i3}$.
2. Para cada valor candidato x (una malla, o búsqueda dicotómica), calcule

$$\hat{p}(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{S_i \leqslant x\}.$$

3. Seleccione $\hat{x}$ tal que $\hat{p}(\hat{x}) \approx 0.95$. Equivalentemente, tome el *cuantil empírico* de orden 0.95 de $\{S_i\}$.

Este procedimiento es el caso de *media muestral* aplicada a la función $g(u, v, w) = \mathbf{1}\{u+v+w \leqslant x\}$ con densidad uniforme en el cubo; ver definición/procedimiento en tus notas.

Código

[6]:
```
# ------------------------------
# 1) Simulación base: S = X1+X2+X3 con Xi ~ Unif(0,15)
# ------------------------------
def simular_sumas(n=1_000_000, seed=123):
    rng = np.random.default_rng(seed)
    X = rng.uniform(0.0, 15.0, size=(n, 3))
    S = X.sum(axis=1)
    S.sort()  # ordenar una vez -> ECDF eficiente/estable
    return S

# ------------------------------
# 2) ECDF y probabilidad estimada P(S <= x)
# ------------------------------
def p_hat_ecdf(S_sorted, x):
```

```
    """
    Devuelve hat{p}(x) = (1/n) * #{ S_i <= x } usando búsqueda binaria
    sobre el arreglo ordenado S_sorted.
    """
    import bisect
    n = len(S_sorted)
    idx = bisect.bisect_right(S_sorted, x)
    return idx / n

# --------------------------------
# 3) Buscar x con p_hat(x) ≈ objetivo (búsqueda dicotómica sobre ECDF)
# --------------------------------
def buscar_x_para_prob(S_sorted, objetivo=0.95, lo=0.0, hi=45.0, tol=1e-4, maxit=60):
    p_lo = p_hat_ecdf(S_sorted, lo)
    p_hi = p_hat_ecdf(S_sorted, hi)
    if objetivo <= p_lo:
        return lo
    if objetivo >= p_hi:
        return hi

    for _ in range(maxit):
        mid = 0.5 * (lo + hi)
        p_mid = p_hat_ecdf(S_sorted, mid)
        if abs(p_mid - objetivo) <= tol:
            return mid
        if p_mid < objetivo:
            lo = mid
        else:
            hi = mid
    return 0.5 * (lo + hi)

# --------------------------------
# 4) Cuantil empírico (ECDF): alternativa directa
# --------------------------------
def cuantil_empirico(S_sorted, q=0.95):
    """
    Devuelve el cuantil empírico de orden q de S_sorted (arreglo ordenado).
    """
    n = len(S_sorted)
    # índice "linear" equivalente a np.quantile con método 'linear'
    pos = q * (n - 1)
    i = int(np.floor(pos))
    frac = pos - i
    if i + 1 < n:
        return (1 - frac) * S_sorted[i] + frac * S_sorted[i + 1]
    else:
        return S_sorted[-1]

# --------------------------------
# Ejecución de ejemplo
# --------------------------------
if __name__ == "__main__":
    # Generar una sola muestra grande -> se usa para todos los x (comparaciones estables)
    S = simular_sumas(n=1_000_000, seed=7)

    # Aproximar x con P(S<=x)=0.95 probando distintos x vía búsqueda dicotómica
    x95_biseccion = buscar_x_para_prob(S, objetivo=0.95)

    # Alternativa: cuantil empírico 0.95 directamente
    x95_empirico = cuantil_empirico(S, q=0.95)

    # Verificación rápida con normal aproximada
    mean_S = 3 * 15 / 2         # = 22.5
    sd_S   = np.sqrt(3 * (15**2) / 12)  # = 7.5
    x95_normal = mean_S + 1.645 * sd_S


    print(f"x (búsqueda sobre ECDF)  ≈ {x95_biseccion:.6f}")
```

```
    print(f"x (cuantil empírico 0.95)≈ {x95_empirico:.6f}")
    print(f"check normal aprox       ≈ {x95_normal:.6f}")
```

```
x (búsqueda sobre ECDF)  ≈ 34.958496
x (cuantil empírico 0.95)≈ 34.954878
check normal aprox       ≈ 34.837500
```

Ejercicio 6

Se tiene el siguiente modelo para el comportamiento de una partícula al tiempo t:

$$X_t \;=\; 3\sqrt{t} + B_t, \qquad t > 0,$$

en donde B_t es una variable aleatoria con distribución $\mathcal{N}(0.2t,\ 0.32t)$. Exprese la probabilidad siguiente como una integral y después apróximela usando simulación:

$$\Pr(X_{10} > 12).$$

Modelo y distribución de X_t

Dado que $B_t \sim \mathcal{N}(0.2t,\ 0.32t)$ y $X_t = 3\sqrt{t} + B_t$, se sigue que

$$X_t \sim \mathcal{N}(\mu(t), \sigma^2(t)), \qquad \mu(t) = 3\sqrt{t} + 0.2t, \quad \sigma^2(t) = 0.32t.$$

Para $t = 10$:

$$\mu = 3\sqrt{10} + 0.2 \cdot 10 = 3\sqrt{10} + 2, \qquad \sigma^2 = 0.32 \cdot 10 = 3.2 = \frac{16}{5}, \qquad \sigma = \sqrt{\frac{16}{5}} = \frac{4}{\sqrt{5}}.$$

Probabilidad como integral y cambio de variable

Como $X_{10} \sim \mathcal{N}(\mu, \sigma^2)$, entonces

$$\Pr(X_{10} > 12) \;=\; \int_{12}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \exp\!\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx.$$

Estandarizando con $z = \dfrac{x-\mu}{\sigma}$ (luego $dx = \sigma\, dz$), el límite inferior se convierte en

$$z_0 = \frac{12-\mu}{\sigma} = \frac{12 - (3\sqrt{10}+2)}{4/\sqrt{5}} = \frac{10 - 3\sqrt{10}}{4/\sqrt{5}} = \frac{\sqrt{5}}{4}\,(10 - 3\sqrt{10}).$$

Por tanto,

$$\Pr(X_{10} > 12) \;=\; \int_{z_0}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-z^2/2}\, dz \;=\; 1 - \Phi(z_0),$$

donde Φ es la CDF de la normal estándar.

Evaluación numérica. Usando $\sqrt{10} \approx 3.16228$ y $\sqrt{5} \approx 2.23607$:

$$\mu \approx 3(3.16228) + 2 = 11.48683, \qquad \sigma = \frac{4}{\sqrt{5}} \approx 1.78885,$$

$$z_0 = \frac{\sqrt{5}}{4}(10 - 3\sqrt{10}) \approx 0.28696, \qquad \Pr(X_{10} > 12) = 1 - \Phi(0.28696) \approx 0.387.$$

Aproximación por simulación

La probabilidad como esperanza de un indicador:

$$\Pr(X_{10} > 12) = \mathbb{E}[\mathbf{1}\{X_{10} > 12\}].$$

Simulamos $B_{10}^{(i)} \sim \mathcal{N}(0.2 \cdot 10,\ 0.32 \cdot 10) = \mathcal{N}(2,\ 3.2)$ e definimos $X_{10}^{(i)} = 3\sqrt{10} + B_{10}^{(i)}$. Con $I_i = \mathbf{1}\{X_{10}^{(i)} > 12\}$, el estimador insesgado es

$$\hat{p} = \frac{1}{n}\sum_{i=1}^{n} I_i, \qquad \widehat{\mathrm{SE}}(\hat{p}) = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}.$$

Un IC aproximado al 95 % es $\hat{p} \pm 1.96\,\widehat{\mathrm{SE}}$.

Código

[7]:
```
# ---------- Analítico ----------
def normal_cdf(z: float) -> float:
    """ (z) usando erf (sin dependencias externas)."""
    return 0.5 * (1.0 + erf(z / sqrt(2.0)))

def prob_analitica():
    """
    P(X10 > 12) con X_t = 3*sqrt(t) + B_t,  B_t ~ N(0.2 t, 0.32 t), t=10.
    Devuelve (p, mu, sigma, z0).
    """
    t = 10.0
    mu = 3.0*sqrt(t) + 0.2*t          # = 3*sqrt(10) + 2
    sigma = sqrt(0.32*t)              # = sqrt(3.2) = 4/sqrt(5)
    z0 = (12.0 - mu) / sigma
    p = 1.0 - normal_cdf(z0)
    return p, mu, sigma, z0

# ---------- Simulación (media muestral) ----------
def prob_mc(n=500_000, seed=7):
    """
    Estima P(X10 > 12) por simulación.
    """
    t = 10.0
    mu_B, var_B = 0.2*t, 0.32*t       # B10 ~ N(2, 3.2)
    sd_B = sqrt(var_B)
    drift = 3.0*sqrt(t)               # 3*sqrt(10)

    rng = np.random.default_rng(seed)
    Z = rng.standard_normal(n)
    B10 = mu_B + sd_B * Z
    X10 = drift + B10

    I = (X10 > 12.0)
    p_hat = float(np.mean(I))
    se = float(np.sqrt(p_hat * (1.0 - p_hat) / n))
    ci = (p_hat - 1.96*se, p_hat + 1.96*se)
    return p_hat, se, ci

# ---------- Ejecución de ejemplo ----------
if __name__ == "__main__":
    p_ex, mu, sigma, z0 = prob_analitica()
    p_mc, se_mc, ci_mc = prob_mc(n=500_000, seed=7)

    print(f"Analítico:  P(X10>12) = {p_ex:.6f}  "
          f"(mu={mu:.6f}, sigma={sigma:.6f}, z0={(z0):.6f})")
    print(f"Monte Carlo: p_hat={p_mc:.6f}, SE={se_mc:.6f}, "
          f"IC95%=[{ci_mc[0]:.6f}, {ci_mc[1]:.6f}]")
```

```
Analítico:  P(X10>12) = 0.387106  (mu=11.486833, sigma=1.788854, z0=0.286869)
Monte Carlo: p_hat=0.387350, SE=0.000689, IC95%=[0.386000, 0.388700]
```

Ejercicio 7

Dentro de la teoría del riesgo se tiene el siguiente modelo para el capital C_t de una compañía aseguradora al tiempo $t \geqslant 0$:

$$C_t \;=\; u + c\,t \;-\; \sum_{j=0}^{N_t} Y_j, \qquad t \geqslant 0,$$

en donde $u \geqslant 0$ y $c > 0$ son constantes, $N_t \sim \text{Poisson}(\lambda t)$ con $\lambda > 0$, y $Y_1, Y_2, \ldots$ son variables aleatorias independientes entre sí e independientes de N_t. Suponga que $Y_0 := 0$ y que $Y_j \sim \text{Exp}(1)$ para $j \geqslant 1$. Suponga también que $u = 1, c = 3$ y $\lambda = 2$. Usando simulación, aproxime:

(a) $\Pr(C_5 < 0)$.

(b) $\Pr(C_5 < 0 \mid Y_1 < 1)$.

Modelo y reexpresión. Para $t = 5$ se tiene $N_5 \sim \text{Poisson}(\lambda t) = \text{Poisson}(10)$ y

$$C_5 \;=\; u + c \cdot 5 \;-\; \sum_{j=1}^{N_5} Y_j \;=\; 16 \;-\; S_{N_5}, \qquad Y_j \overset{iid}{\sim} \text{Exp}(1).$$

Aquí S_{N_5} es la suma compuesta de N_5 reclamaciones exponenciales. Por tanto

$$\Pr(C_5 < 0) \;=\; \Pr\left(S_{N_5} > 16\right) \;=\; \mathbb{E}\big[\mathbf{1}\{S_{N_5} > 16\}\big],$$

lo cual se estima por *media muestral* simulando copias independientes del par (N_5, S_{N_5}).

Condicional. Como $Y_1 \sim \text{Exp}(1)$ es independiente de N_5 y de $\{Y_j\}_{j \geqslant 2}$, se puede escribir

$$\Pr(C_5 < 0 \mid Y_1 < 1) \;=\; \mathbb{E}[\mathbf{1}\{C_5 < 0\} \,|\, Y_1 < 1] \;\approx\; \frac{1}{m} \sum_{i:\, Y_1^{(i)} < 1} \mathbf{1}\{C_5^{(i)} < 0\}.$$

En la simulación se acopla Y_1 dentro de la suma de siniestros (si $N_5 \geqslant 1$, se toma $S_{N_5} = Y_1 + \sum_{j=2}^{N_5} Y_j$; si $N_5 = 0$, entonces $S_0 = 0$). *Nota:* si se desea, puede añadirse la lectura alternativa $\Pr(C_5 < 0 \mid N_5 \geqslant 1,\ Y_1 < 1)$, condicionando además a que ocurra al menos un siniestro antes de $t = 5$.

Código

[8]:
```
def sim_ej7(n=300_000, u=1.0, c=3.0, lam=2.0, t=5.0, seed=7):
    """
    Proceso de riesgo clásico:
      C_t = u + c t - sum_{j=1}^{N_t} Y_j,   N_t ~ Poisson(lam * t),  Y_j ~ Exp(1).

    Devuelve un dict con:
      - p_ruin:           estimación de P(C5 < 0)
      - se_ruin:          error estándar (binomial) de p_ruin
      - p_cond_uncond:    estimación de P(C5 < 0 | Y1 < 1)  (condición 'solo Y1<1')
      - se_cond_uncond:   su error estándar sobre m = # {Y1<1}
      - p_cond_atleast1:  estimación de P(C5 < 0 | N5>=1, Y1 < 1)  (lectura alternativa)
      - se_cond_atleast1: su error estándar sobre m = # {N5>=1, Y1<1}
      - mean_N:           media muestral de N5 (control de calidad; ~ 10)
    """
    rng = np.random.default_rng(seed)
    premium = u + c * t  # u + c t
```

```
    # 1) Número de siniestros
    N = rng.poisson(lam * t, size=n)   # N5

    # 2) Primer siniestro Y1 (definido siempre; independiente de N)
    Y1 = rng.exponential(scale=1.0, size=n)

    # 3) Suma total S = Y1 + Gamma(N-1,1) si N>=1; S=0 si N=0
    S = np.zeros(n)
    mask_ge1 = (N >= 1)
    mask_ge2 = (N >= 2)
    S[mask_ge1] = Y1[mask_ge1]                # incluye Y1
    if np.any(mask_ge2):                      # faltan N-1 ≥ 1 reclamaciones
        k_rest = N[mask_ge2] - 1             # parámetros forma (enteros)
        S[mask_ge2] += rng.gamma(shape=k_rest, scale=1.0)

    # 4) Capital al tiempo t
    C = premium - S

    # (a) P(C5 < 0)
    ruin = (C < 0.0)
    p_ruin = float(np.mean(ruin))
    se_ruin = float(np.sqrt(p_ruin * (1.0 - p_ruin) / n))
    ic95_ruin = (p_ruin - 1.96*se_ruin, p_ruin + 1.96*se_ruin)

    # (b1) Condición "solo Y1<1"
    den1 = (Y1 < 1.0)
    m1 = int(np.count_nonzero(den1))
    p_cond_uncond = float(np.mean(ruin[den1])) if m1 > 0 else np.nan
    se_cond_uncond = float(np.sqrt(p_cond_uncond * (1.0 - p_cond_uncond) / m1)) if m1 > 0 else np.nan
    ic95_cu = (p_cond_uncond - 1.96*se_cond_uncond, p_cond_uncond + 1.96*se_cond_uncond) if m1 > 0 else (np.nan, np.nan)

    # (b2) Condición "Y1<1 y N5>=1"
    den2 = den1 & (N >= 1)
    m2 = int(np.count_nonzero(den2))
    p_cond_atleast1 = float(np.mean(ruin[den2])) if m2 > 0 else np.nan
    se_cond_atleast1 = float(np.sqrt(p_cond_atleast1 * (1.0 - p_cond_atleast1) / m2)) if m2 > 0 else np.nan
    ic95_ca = (p_cond_atleast1 - 1.96*se_cond_atleast1, p_cond_atleast1 + 1.96*se_cond_atleast1) if m2 > 0 else (np.nan,␣
↪np.nan)

    return {
        "n": n,
        "premium": premium,
        "mean_N": float(np.mean(N)),
        "p_ruin": p_ruin, "se_ruin": se_ruin, "IC95_ruin": ic95_ruin,
        "p_cond_uncond": p_cond_uncond, "se_cond_uncond": se_cond_uncond, "IC95_cond_uncond": ic95_cu, "m_uncond": m1,
        "p_cond_atleast1": p_cond_atleast1, "se_cond_atleast1": se_cond_atleast1, "IC95_cond_atleast1": ic95_ca,␣
↪"m_atleast1": m2
    }

# Ejemplo de uso:
res = sim_ej7(n=300_000, seed=7)
for k, v in res.items():
    print(f"{k}: {v}")
```

```
n: 300000
premium: 16.0
mean_N: 9.992766666666666
p_ruin: 0.09878
se_ruin: 0.0005447400346342587
IC95_ruin: (0.09771230953211686, 0.09984769046788315)
p_cond_uncond: 0.07879677337612974
se_cond_uncond: 0.0006190339111531578
IC95_cond_uncond: (0.07758346691026954, 0.08001007984198993)
m_uncond: 189424
p_cond_atleast1: 0.07880301358435977
se_cond_atleast1: 0.0006190808379106504
IC95_cond_atleast1: (0.0775896151420549, 0.08001641202666465)
```

```
m_atleast1: 189409
```

Ejercicio 8

Sea $X \sim \mathcal{N}(0,1)$. Estime $\theta = \Pr(Z > 2)$ mediante muestreo por importancia utilizando la función de densidad exponencial trasladada $f(x)$ que aparece abajo. Tome un tamaño de muestra suficientemente grande para obtener precisión de dos dígitos. Compare con el valor que se obtiene de una tabla de probabilidades de la distribución normal.

$$f(x) = \begin{cases} e^{-(x-2)}, & x > 2, \\ 0, & \text{en otro caso.} \end{cases}$$

Sea $Z \sim \mathcal{N}(0,1)$ y

$$\theta \;=\; \Pr(Z > 2) \;=\; \int_2^\infty \phi(x)\,dx, \qquad \phi(x) = \frac{1}{\sqrt{2\pi}}\, e^{-x^2/2}.$$

Usaremos la densidad exponencial trasladada

$$f(x) = e^{-(x-2)}\,\mathbf{1}_{\{x>2\}} \quad \Longleftrightarrow \quad X \sim 2 + \mathrm{Exp}(1).$$

Como el soporte de f es $(2,\infty)$,

$$\theta = \int_2^\infty \phi(x)\,dx = \int_2^\infty \frac{\phi(x)}{f(x)}\, f(x)\,dx = \mathbb{E}_f[w(X)],$$

donde el *peso de importancia* es

$$w(x) \;=\; \frac{\phi(x)}{f(x)} = \frac{1}{\sqrt{2\pi}} \exp\Big(-\tfrac{x^2}{2} + x - 2\Big) = \phi(x)\, e^{\,x-2}, \qquad x > 2.$$

Por lo tanto, un estimador insesgado de θ es la media muestral de los pesos:

$$\hat\theta_{\text{IS}} = \frac{1}{n}\sum_{i=1}^n w(X_i), \qquad X_i \overset{iid}{\sim} f.$$

Sea $s_w^2 = \frac{1}{n-1}\sum_{i=1}^n \big(w(X_i) - \overline{w}\big)^2$ con $\overline{w} = \hat\theta_{\text{IS}}$. Entonces

$$\widehat{\mathrm{SE}}\big(\hat\theta_{\text{IS}}\big) = \frac{s_w}{\sqrt{n}}, \qquad \text{IC 95\%:} \quad \hat\theta_{\text{IS}} \;\pm\; 1.96\,\widehat{\mathrm{SE}}.$$

El valor "de tabla" es

$$\theta_{\text{tab}} = 1 - \Phi(2) \approx 0.02275013,$$

donde Φ es la CDF normal estándar. También puede compararse con el *Monte Carlo crudo* $\hat\theta_{\text{crudo}} = \frac{1}{n}\sum \mathbf{1}\{Z_i > 2\}$, $Z_i \sim \mathcal{N}(0,1)$, cuyo error estándar es $\sqrt{\hat\theta_{\text{crudo}}\,(1-\hat\theta_{\text{crudo}})/n}$.

Si se desea un *half-width* (radio del IC) no mayor que ε (p. ej. $\varepsilon = 5 \times 10^{-4}$ para estabilizar dos dígitos en 0.02), una aproximación es

$$n \;\gtrsim\; \frac{(1.96)^2\,\mathrm{Var}_f(w)}{\varepsilon^2} \quad \text{(usar un "piloto" para estimar } \mathrm{Var}_f(w) \text{ y redondear hacia arriba).}$$

Algoritmo (pseudocódigo).

1. Fijar n grande (o calcularlo con un piloto).
2. Generar $X_i = 2 + E_i$, $E_i \sim \text{Exp}(1)$, $i = 1, \dots, n$.
3. Calcular $w_i = \phi(X_i)\, e^{X_i - 2}$.
4. Estimar $\widehat{\theta}_{\text{IS}} = \frac{1}{n}\sum w_i$, su $\widehat{\text{SE}} = s_w/\sqrt{n}$ e IC95 %.
5. Comparar con $\theta_{\text{tab}} = 1 - \Phi(2)$ (y, si se desea, con el MC crudo).

Ejercicio 9

Sea (X, Y) un vector aleatorio con distribución Unif$(-1, 1)$ × Unif$(-1, 1)$. Use muestreo condicional para encontrar una aproximación a las probabilidades que aparecen abajo. Calcule el valor exacto de estas probabilidades y compruebe si las aproximaciones obtenidas son razonables.

(a) $\Pr(4X - 2Y > 0)$.

(b) $\Pr(X^2 + Y^2 < 1)$.

(c) $\Pr\left(\frac{X^2}{3} + \frac{Y^2}{2} < 1\right)$.

(a) $\Pr(4X - 2Y > 0)$

El evento es $Y < 2X$. Condicionando en $X = x$ y usando que $Y \sim$ Unif$(-1, 1)$,

$$p(x) = \Pr(Y < 2x \mid X = x) = \begin{cases} 0, & x \leqslant -\frac{1}{2}, \\ \dfrac{2x+1}{2}, & -\frac{1}{2} < x < \frac{1}{2}, \\ 1, & x \geqslant \frac{1}{2}. \end{cases}$$

Entonces

$$\Pr(4X - 2Y > 0) = \int_{-1}^{1} p(x)\,\frac{1}{2}\,dx = \frac{1}{2}\left[\int_{-1/2}^{1/2} \frac{2x+1}{2}\,dx + \int_{1/2}^{1} 1\,dx\right].$$

Cálculo:

$$\int_{-1/2}^{1/2} \frac{2x+1}{2}\,dx = \frac{1}{4}\int_{-1/2}^{1/2} (2x+1)\,dx = \frac{1}{4}\Big[x^2 + x\Big]_{-1/2}^{1/2} = \frac{1}{4}\Big(\tfrac{3}{4} - (-\tfrac{1}{4})\Big) = \frac{1}{4},$$

$$\int_{1/2}^{1} 1\,dx = \frac{1}{2}, \qquad \Rightarrow \quad \Pr(4X - 2Y > 0) = \frac{1}{2}\left(\frac{1}{4} + \frac{1}{2}\right) = \boxed{\tfrac{1}{2}}.$$

(b) $\Pr(X^2 + Y^2 < 1)$

Dado $X = x$, el evento es $|Y| < \sqrt{1 - x^2}$ (si $|x| < 1$; si $|x| \geqslant 1$ la probabilidad es 0). Como $Y \sim$ Unif$(-1, 1)$, la longitud del intervalo admisible para Y es $2\sqrt{1 - x^2}$, y por tanto

$$p(x) = \Pr(|Y| < \sqrt{1 - x^2} \mid X = x) = \frac{2\sqrt{1 - x^2}}{2} = \sqrt{1 - x^2}, \qquad |x| < 1.$$

Luego

$$\Pr(X^2 + Y^2 < 1) = \int_{-1}^{1} \sqrt{1 - x^2}\,\frac{1}{2}\,dx = \frac{1}{2}\int_{-1}^{1} \sqrt{1 - x^2}\,dx.$$

Usando una primitiva estándar $\int \sqrt{1 - x^2}\,dx = \frac{1}{2}\big(x\sqrt{1 - x^2} + \arcsin x\big) + C$,

$$\int_{-1}^{1} \sqrt{1 - x^2}\,dx = \Big[\tfrac{1}{2}\big(x\sqrt{1 - x^2} + \arcsin x\big)\Big]_{-1}^{1} = \tfrac{1}{2}\big(\tfrac{\pi}{2} - (-\tfrac{\pi}{2})\big) = \frac{\pi}{2}.$$

Así,

$$\Pr(X^2+Y^2<1)=\frac{1}{2}\cdot\frac{\pi}{2}=\boxed{\frac{\pi}{4}}\approx 0.7854.$$

(c) $\Pr(\frac{X^2}{3}+\frac{Y^2}{2}<1)$

Dado $X=x$, la condición es $|Y|<\sqrt{2(1-\frac{x^2}{3})}$. Para todo $|x|<1$,

$$1-\frac{x^2}{3}>\frac{2}{3}\;\Rightarrow\;2\left(1-\frac{x^2}{3}\right)>\frac{4}{3}\;\Rightarrow\;\sqrt{2\left(1-\frac{x^2}{3}\right)}>\sqrt{\frac{4}{3}}>1.$$

Como $Y\in(-1,1)$, la desigualdad se satisface *siempre*; por lo tanto $p(x)\equiv 1$ y

$$\Pr\left(\frac{X^2}{3}+\frac{Y^2}{2}<1\right)=\int_{-1}^{1}1\cdot\frac{1}{2}\,dx=\boxed{1}.$$

$$\boxed{\Pr(4X-2Y>0)=\tfrac{1}{2}},\qquad\boxed{\Pr(X^2+Y^2<1)=\tfrac{\pi}{4}},\qquad\boxed{\Pr(\tfrac{X^2}{3}+\tfrac{Y^2}{2}<1)=1}.$$

Código

[9]:
```
# ---------------------------
# Probabilidades exactas
# ---------------------------
def exact_a(): return 0.5
def exact_b(): return np.pi / 4.0
def exact_c(): return 1.0

# ---------------------------
# p(x) condicionales
# ---------------------------
def p_a_of_x(x):
    # piecewise: 0, (2x+1)/2, 1
    out = np.zeros_like(x, dtype=float)
    mask_mid = (x > -0.5) & (x < 0.5)
    mask_hi  = (x >= 0.5)
    out[mask_mid] = (2.0 * x[mask_mid] + 1.0) / 2.0
    out[mask_hi]  = 1.0
    return out

def p_b_of_x(x):
    # sqrt(1 - x^2), safe
    val = 1.0 - x*x
    val[val < 0] = 0.0
    return np.sqrt(val)

def p_c_of_x(x):
    # always 1 on (-1,1)
    return np.ones_like(x, dtype=float)

# ---------------------------
# Estimadores por muestreo condicional
# ---------------------------
def cond_estimator(p_of_x, n=200_000, seed=7):
    rng = np.random.default_rng(seed)
    x = rng.uniform(-1.0, 1.0, size=n)      # X ~ Unif(-1,1)
    p_vals = p_of_x(x)                      # p(X)
```

```
    theta_hat = float(np.mean(p_vals)) / 1.0  # E[p(X)] (f_X=1/2 se refleja en p(x) ya definido? -> p(x) arriba ya es prob␣
↪condicional; su promedio directo es la prob, porque usamos media muestral sobre X ~ Unif(-1,1) y la densidad 1/2 se␣
↪incorpora en la esperanza -> tomar mean(p_vals) está bien si x ~ Unif(-1,1) con peso uniforme; estrictamente E[p(X)] =␣
↪∫ p(x) f_X dx; discretamente es el promedio simple al muestrear de f_X)
    # Error estándar (de la media de p(X))
    se = float(np.std(p_vals, ddof=1) / np.sqrt(n))
    ci = (theta_hat - 1.96 * se, theta_hat + 1.96 * se)
    return theta_hat, se, ci

# ---------------------------
# Estimadores “crudos” para comparar (opcional)
# ---------------------------
def crude_estimator(event_fn, n=200_000, seed=7):
    rng = np.random.default_rng(seed)
    x = rng.uniform(-1.0, 1.0, size=n)
    y = rng.uniform(-1.0, 1.0, size=n)
    I = event_fn(x, y).astype(float)
    p = float(np.mean(I))
    se = float(np.sqrt(p * (1 - p) / n))
    ci = (p - 1.96 * se, p + 1.96 * se)
    return p, se, ci

def event_a(x, y): return (4.0*x - 2.0*y) > 0.0
def event_b(x, y): return (x*x + y*y) < 1.0
def event_c(x, y): return (x*x/3.0 + y*y/2.0) < 1.0

# ---------------------------
# Ejemplo de uso
# ---------------------------
if __name__ == "__main__":
    n = 200_000

    # (a)
    th_a, se_a, ci_a = cond_estimator(p_a_of_x, n=n, seed=1)
    cr_a, secr_a, ccr_a = crude_estimator(event_a, n=n, seed=1)
    print("(a) cond:", th_a, "SE:", se_a, "IC95:", ci_a, "| exacto:", exact_a())
    print("    crudo:", cr_a, "SE:", secr_a, "IC95:", ccr_a)

    # (b)
    th_b, se_b, ci_b = cond_estimator(p_b_of_x, n=n, seed=2)
    cr_b, secr_b, ccr_b = crude_estimator(event_b, n=n, seed=2)
    print("(b) cond:", th_b, "SE:", se_b, "IC95:", ci_b, "| exacto:", exact_b())
    print("    crudo:", cr_b, "SE:", secr_b, "IC95:", ccr_b)

    # (c)
    th_c, se_c, ci_c = cond_estimator(p_c_of_x, n=n, seed=3)
    cr_c, secr_c, ccr_c = crude_estimator(event_c, n=n, seed=3)
    print("(c) cond:", th_c, "SE:", se_c, "IC95:", ci_c, "| exacto:", exact_c())
    print("    crudo:", cr_c, "SE:", secr_c, "IC95:", ccr_c)
```

```
(a) cond: 0.49944514018736547 SE: 0.0009129076713279941 IC95:
(0.4976558411515626, 0.5012344392231683) | exacto: 0.5
    crudo: 0.49887 SE: 0.0011180311335110486 IC95: (0.4966786589783183,
0.5010613410216816)
(b) cond: 0.7847524382953979 SE: 0.0004990404544118926 IC95:
(0.7837743190047506, 0.7857305575860452) | exacto: 0.7853981633974483
    crudo: 0.785135 SE: 0.0009184172030591543 IC95: (0.7833349022820041,
0.786935097717996)
(c) cond: 1.0 SE: 0.0 IC95: (1.0, 1.0) | exacto: 1.0
    crudo: 1.0 SE: 0.0 IC95: (1.0, 1.0)
```

Ejercicio 10

Sea (X, N) un vector aleatorio discreto con función de probabilidad

$$f(x,n) \;=\; \frac{3}{10}\left(\tfrac{1}{2}\right)^{nx}, \qquad n = 1,2;\; x = 0,1,2,\ldots$$

y suponga que X_1, X_2 son dos copias independientes de X (independientes entre sí y de N).

(a) Usando muestreo condicional sobre N, encuentre una aproximación a la esperanza de

$$S = \sum_{i=1}^{N} X_i.$$

(b) Encuentre el valor exacto de $\mathbb{E}(S)$ y compruebe si la aproximación obtenida es razonable.

La marginal de N se obtiene con series geométricas:

$$\Pr(N=n) = \sum_{x\geqslant 0} f(x,n) = \frac{3}{10}\sum_{x\geqslant 0}\left(\tfrac{1}{2}\right)^{nx} = \frac{3}{10}\,\frac{1}{1-(1/2)^n} = \begin{cases} \frac{3}{5}, & n=1, \\ \frac{2}{5}, & n=2. \end{cases}$$

La condicional de X dado $N=n$ es

$$\Pr(X=x \mid N=n) = \frac{f(x,n)}{\Pr(N=n)} = \left(1-\left(\tfrac{1}{2}\right)^n\right)\left(\tfrac{1}{2}\right)^{nx}, \qquad x = 0,1,\ldots$$

que es *geométrica* en $\{0,1,\ldots\}$ con parámetro $p_n = (1/2)^n$ (en la convención "número de fallas antes del primer éxito"). Así,

$$\mathbb{E}[X \mid N=n] = \frac{p_n}{1-p_n} = \frac{(1/2)^n}{1-(1/2)^n}, \quad \Rightarrow \quad \mathbb{E}[X \mid N=1] = 1,\;\; \mathbb{E}[X \mid N=2] = \tfrac{1}{3}.$$

2) Media de X y de N. Por la ley de la esperanza total,

$$\mu_X := \mathbb{E}[X] = \sum_n \Pr(N=n)\,\mathbb{E}[X \mid N=n] = \frac{3}{5}\cdot 1 + \frac{2}{5}\cdot\frac{1}{3} = \boxed{\frac{11}{15}}.$$

La media de N (dos puntos):

$$\mu_N := \mathbb{E}[N] = 1\cdot\frac{3}{5} + 2\cdot\frac{2}{5} = \boxed{\frac{7}{5}}.$$

3)Valor exacto de $\mathbb{E}[S]$. Como X_i son i.i.d. con ley X e independientes de N,

$$\mathbb{E}[S \mid N] = N\,\mu_X \quad \Longrightarrow \quad \mathbb{E}[S] = \mathbb{E}[\mathbb{E}[S \mid N]] = \mu_X\,\mathbb{E}[N] = \boxed{\frac{11}{15}\cdot\frac{7}{5} = \frac{77}{75} \approx 1.026\overline{6}}.$$

4) Estimación por *muestreo condicional* sobre N. Para aproximar $\mathbb{E}[S]$ por simulación con menor varianza que el método crudo:

1. Generar $N_1, \ldots, N_m$ i.i.d. con $\Pr(N=1) = 3/5$, $\Pr(N=2) = 2/5$.
2. Usar $\mu_X = \mathbb{E}[X] = 11/15$ (o estimarla con un piloto independiente).

3. Estimador CMC:

$$\hat{\mu}_S^{(\mathrm{cond})} = \frac{1}{m} \sum_{k=1}^{m} N_k \, \mu_X,$$

con error estándar $\widehat{\mathrm{SE}} = \mathrm{SD}(N\,\mu_X)/\sqrt{m}$ e intervalo 95 %: $\hat{\mu}_S^{(\mathrm{cond})} \pm 1.96\,\widehat{\mathrm{SE}}$.

Este estimador implementa $\mathbb{E}[S] = \mathbb{E}\{\mathbb{E}[S \mid N]\}$ y evita simular todos los X_i en cada réplica, logrando *reducción de varianza*.

Código

[10]:

```
pN1, pN2 = 3/5, 2/5
E_N = 1*pN1 + 2*pN2                 # = 7/5
# E[X] por series: (3/10) * [ sum x (1/2)^x + sum x (1/4)^x ]
E_X = (3/10) * ((0.5) / (1-0.5)**2 + (0.25) / (1-0.25)**2)  # = 11/15
E_S_exact = E_N * E_X                                       # = 77/75

def sample_N(rng, size):
    """Muestra N in {1,2} con P(N=1)=3/5, P(N=2)=2/5."""
    u = rng.random(size)
    return np.where(u < pN1, 1, 2)

def sample_X_given_n(rng, n):
    """
    Muestra X | N=n. Condicional es geométrica en {0,1,...} con p_n=1-(1/2)^n.
    Numpy geometric devuelve {1,2,...} con media 1/p -> restamos 1.
    """
    pn = 1 - 0.5**n
    return rng.geometric(p=pn) - 1

def sample_X_marginal(rng, size):
    """
    Muestra X de su marginal P(X=x) = (3/10)[2^{-x} + 4^{-x}].
    Se usa el truco de mezcla: elegir N' ~ {1,2} con (3/5, 2/5) y luego X|N'.
    """
    Nprime = sample_N(rng, size)
    return sample_X_given_n(rng, Nprime)

# ---------- Simulación condicional sobre N ----------
def estimate_ES_conditional(m=200_000, seed=7):
    rng = np.random.default_rng(seed)
    N = sample_N(rng, m)
    mu_X = E_X                        # usar el valor exacto (puedes reemplazar por estimado si quieres)
    S_hat = N * mu_X
    mu_hat = float(np.mean(S_hat))
    # Error estándar de la media de N*mu_X:
    se = float(np.std(S_hat, ddof=1) / np.sqrt(m))
    ci = (mu_hat - 1.96*se, mu_hat + 1.96*se)
    return mu_hat, se, ci

# ---------- Simulación "cruda" (para comparar) ----------
def estimate_ES_crude(m=200_000, seed=8):
    rng = np.random.default_rng(seed)
    N = sample_N(rng, m)
    S = np.zeros(m, dtype=float)
    # casos N=1 y N=2 (únicos posibles)
    mask1 = (N == 1)
    mask2 = ~mask1
    S[mask1] = sample_X_marginal(rng, np.sum(mask1))
    if np.any(mask2):
        X1 = sample_X_marginal(rng, np.sum(mask2))
        X2 = sample_X_marginal(rng, np.sum(mask2))
        S[mask2] = X1 + X2
    mu_hat = float(np.mean(S))
    se = float(np.std(S, ddof=1) / np.sqrt(m))
    ci = (mu_hat - 1.96*se, mu_hat + 1.96*se)
```

```
    return mu_hat, se, ci

# ---------- Demostración ----------
if __name__ == "__main__":
    m = 300_000
    mu_c, se_c, ci_c = estimate_ES_conditional(m=m, seed=7)
    mu_r, se_r, ci_r = estimate_ES_crude(m=m, seed=8)

    print(f"E[S] exacta             = {E_S_exact:.8f}  (= 77/75)")
    print(f"Condicional sobre N     = {mu_c:.8f}   SE={se_c:.6f}  IC95={ci_c}")
    print(f"Crudo (simular X_i)     = {mu_r:.8f}   SE={se_r:.6f}  IC95={ci_r}")
    if se_c > 0:
        print(f"Reducción de varianza ≈ {(se_r**2)/(se_c**2):.2f}x (SE_crudo^2 / SE_cond^2)")
```

```
E[S] exacta             = 1.02666667  (= 77/75)
Condicional sobre N     = 1.02729489   SE=0.000656  IC95=(1.0260088398921172,
1.02858093788566)
Crudo (simular X_i)     = 1.03039000   SE=0.002725  IC95=(1.0250482811313377,
1.035731718868662)
Reducción de varianza ≈ 17.25x (SE_crudo^2 / SE_cond^2)
```

Ejercicio 11

Defina la variable aleatoria $S := X^N$, en donde X y N son independientes y son tales que $X \sim \text{Unif}(0,1)$ y $N \sim \text{Unif}\{1,\dots,10\}$.

(a) Usando muestreo condicional sobre N, encuentre una aproximación para $\mathbb{E}(S)$.

(b) Encuentre el valor exacto de $\mathbb{E}(S)$ y compruebe si la aproximación obtenida es razonable.

Idea (muestreo condicional). Como $S = X^N$ y $X \perp N$,

$$\mathbb{E}(S) = \mathbb{E}\big[\mathbb{E}(X^N \mid N)\big] = \mathbb{E}\big[\mathbb{E}(X^n)\big]_{n=N}.$$

Si $X \sim \text{Unif}(0,1)$, entonces $\mathbb{E}(X^n) = \int_0^1 x^n\,dx = \frac{1}{n+1}$. Por lo tanto,

$$\mathbb{E}(S) = \mathbb{E}\Big(\frac{1}{N+1}\Big) = \frac{1}{10}\sum_{n=1}^{10}\frac{1}{n+1} = \frac{1}{10}\sum_{k=2}^{11}\frac{1}{k} = \frac{H_{11}-1}{10},$$

donde $H_m = \sum_{k=1}^{m}\frac{1}{k}$ es el número armónico. Numéricamente,

$$H_{11} = 1 + \frac{1}{2} + \cdots + \frac{1}{11} \approx 3.019877344, \qquad \Rightarrow \quad \mathbb{E}(S) \approx \frac{3.019877344 - 1}{10} = 0.2019877344.$$

Estimación por CMC (Conditional Monte Carlo). En lugar de simular X y elevarlo a un exponente aleatorio, basta simular N y usar $\mathbb{E}(X^N \mid N = n) = \frac{1}{n+1}$:

$$\hat{\mu}_{\text{CMC}} = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{N_i+1}, \qquad N_i \overset{iid}{\sim} \text{Unif}\{1,\dots,10\}.$$

Este estimador es insesgado y típicamente tiene menor varianza que el "crudo" $\hat{\mu}_{\text{crudo}} = \frac{1}{m}\sum_{i=1}^{m} X_i^{N_i}$, pues sustituye X^N por su esperanza condicional. El error estándar de $\hat{\mu}_{\text{CMC}}$ es $\widehat{\text{SE}} = \text{SD}\big(\frac{1}{N+1}\big)/\sqrt{m}$, y un IC95 % es $\hat{\mu}_{\text{CMC}} \pm 1.96\,\widehat{\text{SE}}$.

Código

[11]:
```python
def exact_E_S():
    # (H_11 - 1)/10
    H11 = sum(1.0/k for k in range(1, 12))
    return (H11 - 1.0) / 10.0

def E_S_cmc(m=100_000, seed=7):
    rng = np.random.default_rng(seed)
    N = rng.integers(1, 11, size=m)          # Unif{1,...,10}
    vals = 1.0 / (N + 1.0)                   # E[X^N | N]
    mu = float(np.mean(vals))
    se = float(np.std(vals, ddof=1) / np.sqrt(m))
    ci = (mu - 1.96*se, mu + 1.96*se)
    return mu, se, ci

def E_S_crudo(m=100_000, seed=8):
    rng = np.random.default_rng(seed)
    N = rng.integers(1, 11, size=m)
    X = rng.uniform(0.0, 1.0, size=m)
    S = X ** N
    mu = float(np.mean(S))
```

```
    se = float(np.std(S, ddof=1) / np.sqrt(m))
    ci = (mu - 1.96*se, mu + 1.96*se)
    return mu, se, ci

if __name__ == "__main__":
    exact = exact_E_S()
    mu_cmc, se_cmc, ci_cmc = E_S_cmc(m=200_000, seed=7)
    mu_raw, se_raw, ci_raw = E_S_crudo(m=200_000, seed=7)

    print(f"E[S] exacta      = {exact:.10f}")
    print(f"CMC (solo N)     = {mu_cmc:.10f}  SE={se_cmc:.6f}  IC95={ci_cmc}")
    print(f"Crudo (X^N)      = {mu_raw:.10f}  SE={se_raw:.6f}  IC95={ci_raw}")
    if se_cmc > 0:
        print(f"Reducción var. ≈ {(se_raw**2)/(se_cmc**2):.2f}x (SE_crudo^2/SE_CMC^2)")
```

```
E[S] exacta      = 0.2019877345
CMC (solo N)     = 0.2017700258  SE=0.000273  IC95=(0.2012342275226516,
0.20230582406464995)
Crudo (X^N)      = 0.2015832854  SE=0.000620  IC95=(0.20036824333333011,
0.2027983275502032)
Reducción var. ≈ 5.14x (SE_crudo^2/SE_CMC^2)
```