Métodos de MonteCarlo

Curso 2023

Obligatorio 3 - Ejercicio 6.2

Grupo 05

Juan Manuel Varela

Nicolás Farías

Contenido

Descripción del Problema **3**
Solución **4**
Parte a 4
Parte b 4
Parte c 4
Anexo **5**
Detalles Técnicos 5
Log de Ejecución 5
Código fuente 5

Descripción del Problema

Ejercicio 6.2: [en grupo]

Problema: se desea estimar la integral de la función $x_1 x_2^2 x_3^3 x_4^4 x_5^5$ sobre el hipercubo J^m de dimensión m = 5.

- Parte a: revisar los códigos preparados para el ejercicio 6.1, elegir uno de ellos como punto de partida. Sobre esa base, modificarlo para realizar cálculo por Monte Carlo de la integral planteada en el ejercicio 6.2. realizar 10^6 replicaciones y estimar el valor de ζ. Calcular analíticamente el valor exacto de la integral.

- Parte b: en base al valor estimado en la parte a, calcular el número de replicaciones necesario para obtener un error menor a 10^{-4} (con nivel de confianza 0.95).

- Parte c: Decimos que un intervalo de confianza cubre el valor exacto cuando este último pertenece al intervalo.
  Realizar L = 500 experimentos con semillas diferentes, cada uno consistente en estimar por Monte Carlo con el nro. de replicaciones de la parte b el valor de la integral, así como intervalos de confianza de nivel 0.9, 0.95 y 0.99. Para cada nivel de confianza, calcular el nivel de cobertura empírico (en qué porcentaje de los 500 experimentos el intervalo de confianza calculado cubrió el valor exacto).
  Discutir los resultados, comparando la cobertura empírica con la especificada.

Solución

Parte a

No se detectaron errores en los códigos revisados. Al estar implementados en el mismo lenguaje (Python) y utilizando la misma biblioteca para generar los números pseudoaleatorios, se pudo ejecutar los dos con la misma semilla y se obtuvieron resultados idénticos.
Se adaptó el código para calcular la integral planteada. Luego de ejecutarlo con 10^6 replicaciones, se obtuvo un valor estimado de la integral de 0.001393.
Si calculamos la integral analíticamente obtenemos:

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 x_1 x_2^2 x_3^3 x_4^4 x_5^5 dx_1 dx_2 dx_3 dx_4 dx_5 = \frac{1}{2}.\frac{1}{3}.\frac{1}{4}.\frac{1}{5}.\frac{1}{6} = 0.001389$$

Parte b

Aplicamos la fórmula para calcular la cantidad de replicaciones:

$$\dot{n}_N(\epsilon, \delta) = \lceil \left(\Phi^{-1}\dot{(}1 - \delta/2)\right)^2 \dot{\sigma}^2_{n'}/\epsilon^2 \rceil$$

La expresión de Python utilizada para el cálculo fue:

- math.ceil(scipy.stats.norm.ppf(1 - delta / 2)**2 * varianza_funcion / epsilon**2)

Utilizamos el estimador de la varianza de la función calculado con la ejecución de la parte a. El resultado es 36046.

Parte c

Ejecutamos los 500 experimentos utilizando como semilla los números del 0 al 499.
Se muestra a continuación una tabla con los niveles de cobertura empírico que obtuvimos

Cobertura especificada	Cobertura empírica
0.9	0.922000
0.95	0.954000
0.99	0.990000

Se puede observar que en este caso el nivel de cobertura empírico igualó o superó al nivel de cobertura especificada para los tres intervalos. Es importante tener presente que una elección distinta de semillas puede llevar a resultados levemente diferentes para esta cantidad de ejecuciones.

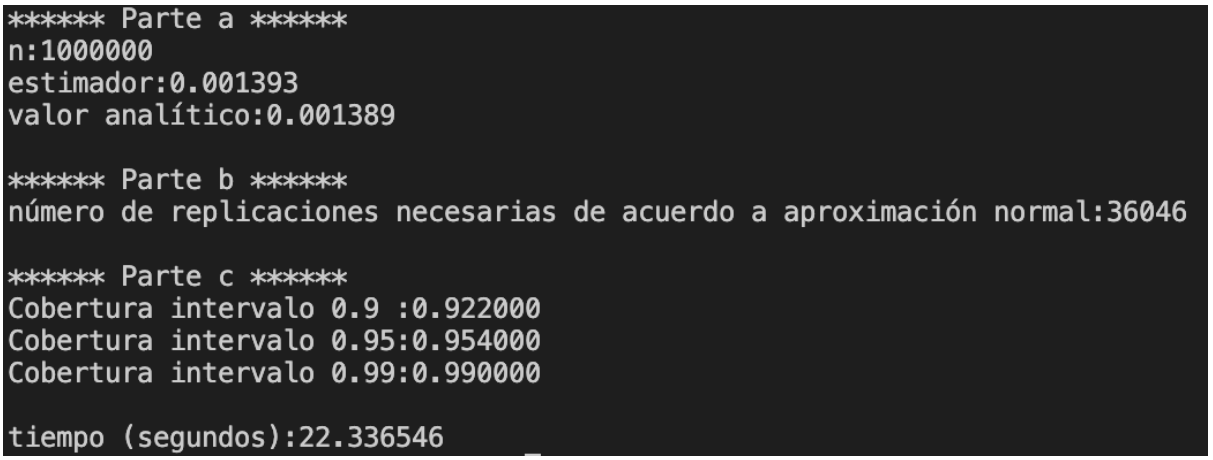
Anexo

Detalles Técnicos

El código se implementó en lenguaje Python (versión 3.11.2).
Se ejecutaron los programas en una notebook con las siguientes características: MacBook Pro 2017, Procesador 3.1 GHz Quad-Core Intel Core i7, Memoria RAM 16 GB 2133 MHz LPDDR3, Sistema Operativo macOS Ventura 13.2.1

Log de Ejecución

Se muestra a continuación una captura de pantalla de la ejecución del programa.



Código fuente

```
import random
import math
import time
import scipy


def fi(x):
    """Devuelve el resultado de evaluar la función en un punto."""
    return x[0] * x[1]**2 * x[2]**3 * x[3]**4 * x[4]**5
```

```
def monte_carlo(semilla, n):
    """Cálculo de integral por monte carlo."""

    # inicializa generador de números con semilla
    random.seed(semilla)

    # inicialización
    S = 0
    T = 0

    for j in range(1, n + 1):
        x = [random.random() for _ in range(5)]
        res = fi(x)
        if j > 1:
            T = T + (1 - 1/j) * (res - S / (j - 1))**2
        S += res

    estimador = S / n
    varianza_funcion = T / (n - 1)
    varianza_estimador = varianza_funcion / n

    return estimador, varianza_funcion

# Parte a

# tiempo inicial
start = time.time()

n = 1000000
estimador, varianza_funcion = monte_carlo(12345, n)
valor_analítico = 1 / (2 * 3 * 4 * 5 * 6)

print('****** Parte a ******')
print(f'n:{n}')
print(f'estimador:{estimador:f}')
print(f'valor analítico:{valor_analítico:f}')
print()

# Parte b
```

```
delta = 0.05
epsilon = 0.0001
num_replicaciones = math.ceil(scipy.stats.norm.ppf(1 - delta / 2)**2
* varianza_funcion / epsilon**2)

print('****** Parte b ******')
print(f'número de replicaciones necesarias de acuerdo a aproximación
normal:{num_replicaciones}')
print()

# Parte c

print('****** Parte c ******')
L = 500
n = num_replicaciones
deltas = [0.1, 0.05, 0.01]
coberturas = [0, 0, 0]

for i in range(L):
    estimador, varianza_funcion = monte_carlo(12346 + i,
num_replicaciones)

    # Cálculo de intervalo de confianza empleando aproximación
normal
    for j in range(3):
        in1 = estimador - scipy.stats.norm.ppf(1 - deltas[j] / 2) *
(varianza_funcion / n) ** 0.5
        in2 = estimador + scipy.stats.norm.ppf(1 - deltas[j] / 2) *
(varianza_funcion / n) ** 0.5
        if valor_analítico >= in1 and valor_analítico <= in2:
            coberturas[j] += 1

# tiempo final
end = time.time()

print(f'Cobertura intervalo 0.9 :{coberturas[0]/L:f}')
print(f'Cobertura intervalo 0.95:{coberturas[1]/L:f}')
print(f'Cobertura intervalo 0.99:{coberturas[2]/L:f}')
print()
print(f'tiempo (segundos):{end - start:f}')
```