INFORME TAREA 6

Benjamín Briceño
RUT: 20.653.153-3
Github: @Benbriel

1. Problema 1: Integración de Monte Carlo

1.1. Introducción

Esta sección de la tarea pretende calcular el volumen formado por el interior de un tubo en T ("pipe tee") de dimensiones como en la Figura 1, utilizando un algoritmo de integración de Monte Carlo de primera forma, es decir, con una distribución uniforme.

Figura 1: Tubo en T de dimensiones indicadas. Su volumen consta de dos cilindros perpendiculares, donde el cilindro corto "sale"desde la mitad del grande

La integración de Monte Carlo estima que para un número N grande de puntos n-dimensionales aleatorios uniformes distribuidos en un espacio, se puede estimar el valor de un volumen específico f, o una integral, tal que

$$\int f dV \approx V\langle f\rangle \pm V\sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}} \tag{1}$$

Donde $\langle f\rangle = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$ es la media aritmética de f y $\langle f^2\rangle = \frac{1}{N}\sum_{i=1}^{N} f^2(x_i)$ la media de f^2. Así, el primer término es el volumen estimado, y el segundo una medida de la incertidumbre asociada.

Además, se estimó el volumen de la figura para distintas cantidades de puntos, y se estimó la incertidumbre asociada a la medición iterando el cálculo de 100 volúmenes por cada n y calculando su desviación estándar. Por último, se estudió numéricamente la tendencia de la incertidumbre al aumentar el número de puntos aleatorios evaluados por cada volumen.

1.2. Desarrollo

Primero, se encerró la Figura 1 en una caja de $V = 4 \cdot 6 \cdot 2$ cm^3. Se escogió como origen del eje de coordenadas el punto central del cilindro largo, y se estableció $\hat{x}$ en la dirección del cilindro pequeño, hacia abajo, $\hat{y}$ hacia la derecha y paralelo al cilindro grande, y $\hat{z}$ hacia arriba. Con esto, se creó una función de `Python` que evalúa un punto (x, y, z) y retorna un `bool` si el punto pertenece a la figura o no, equivalente a $f(x_i)$ de la ecuación 1, tal que

```
def is_in_pipe(x, y, z):
    tubo_largo = abs(y) <= 3 and x**2 + z**2 <= 1
    tubo_corto = 0 <= x <= 3 and y**2 + z**2 <= 1
    return tubo_largo or tubo_corto
```

Luego, se creó una distribución aleatoria uniforme de $n = 1 \cdot 10^5$ puntos en V utilizando la función `uniform` del módulo `numpy.random`. Esta función utiliza un generador de números pseudo-aleatorios para simular las propiedades de los números aleatorios, pero mantener la consistencia al entregar la misma serie de números cada vez que se compile el código.

Tras esto, se evaluaron los puntos en `is_in_pipe`, que entregó un array de unos o ceros. Cabe destacar la utilización del módulo `numba` y la función `njit` como decorador de algunas funciones del código, que permitió acortar el tiempo de compilación y así escoger un n órdenes de magnitud mayor que lo posible en un tiempo razonable. El módulo `numba` traduce una función que utilice arrays de `NumPy` y loops `for` a lenguaje de máquina, lo que reduce hasta 100 veces los tiempos de compilación.

Los resultados obtenidos para $n = 10^5$ fueron

$$V_{fig} = 25.603 \pm 0.077 \tag{2}$$

Esta incertidumbre, si bien es relativamente grande, es de esperar para los algoritmos de integración de Monte Carlo. Finalmente, se iteró el cálculo del volumen 100 veces por cada número de puntos n, y se calculó su media $\bar{V_n}$ y desviación estándar σ_n. Además, se realizó el proceso con múltiples valores de n menores que 10^5, con tal de visualizar la tendencia de la incertidumbre asociada. Para cada cálculo de un volumen se utilizó una serie aleatoria distinta.

Se puede notar que las desviaciones de la Figura 2 se comportan $\propto 1/\sqrt{n}$. Esto fue realizado sin determinar la incertidumbre de la ecuación 1, por lo que se demuestra que el comportamiento de la incertidumbre es equivalente a la desviación estándar de un conjunto de volúmenes para un n. Además, se graficó el valor de los volúmenes promedio para cada n.

La Figura 3 muestra la convergencia de los volúmenes hacia el valor real del tubo en T. Cabe destacar que la desviación estándar es grande para n pequeños, sin embargo, como el volumen estimado se obtiene promediando 100 datos, la separación de los datos pierde importancia y se puede aproximar bien el volumen real para n pequeños.

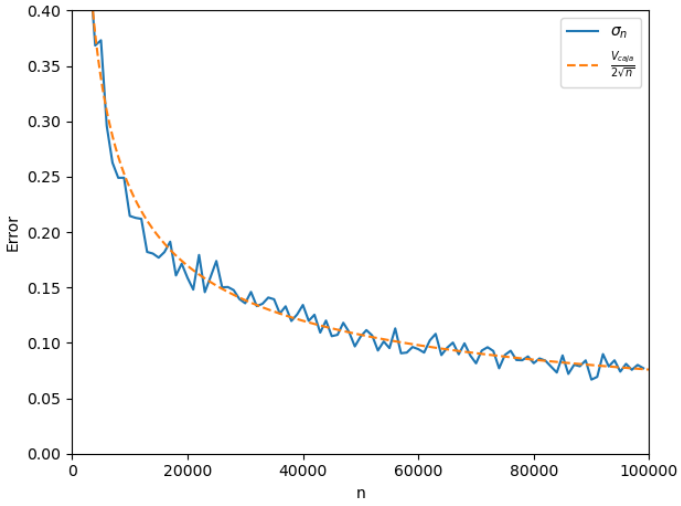

Figura 2: Desviación estándar de 100 volúmenes calculados con n puntos y función de muestra. Las funciones no convergen necesariamente al mismo límite, sin embargo, se visualiza el orden de decrecimiento del error.

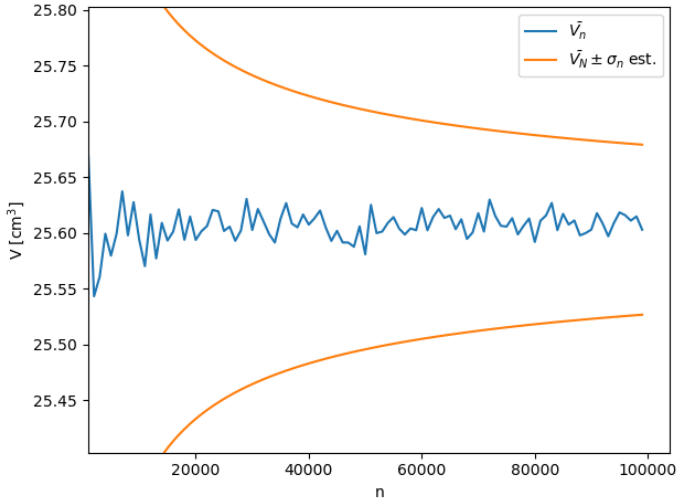

Figura 3: Volúmenes promedio para cada n. Además, se incluye el error asociado de muestra de la Figura 2 para visualizar la desviación de los volúmenes. Se utilizó $\bar{V}_N = V_{fig}$ como punto central.

1.3. Discusión y Conclusiones

Se obtuvo que la desviación estándar de un grupo de volúmenes calculados con n puntos se comporta proporcional a $1/\sqrt{n}$, lo que es equivalente a la fórmula de 1. Esto es idéntico a concluir que σ_n converge a un valor al aumentar el número de volúmenes calculados, que en este caso fue 100.

Se puede concluir que el método de Monte Carlo fue eficaz para aproximar el valor del volumen

de la Figura 1. Sin embargo, para obtener una mayor precisión, el error asociado disminuye $\propto 1/\sqrt{n}$, por lo que si se utiliza un n 10 veces mayor, el error será únicamente $1/\sqrt{10}$ veces el anterior. Esto requiere mucho poder computacional, y a veces puede no ser efectivo.

2. Problema 2: Algoritmo de Metrópolis

2.1. Introducción

Esta parte de la tarea pretende obtener datos de una muestra aleatoria de una distribución de probabilidad dada, utilizando el algoritmo de Metrópolis. Para esto, se tiene una distribución de probabilidad de densidad

$$W(x) = x^{\alpha-1}e^{-\beta x} \tag{3}$$

El algoritmo de Metrópolis toma puntos de una distribución aleatoria propuesta, y los acepta como parte de la muestra si $W(x_p) > W(x_n) \cdot U(0,1)$, con $U(0,1)$ distribución aleatoria uniforme entre 0 y 1. Los puntos aceptados se consideran para la siguiente iteración del algoritmo, y los rechazados no afectan a la iteración.

Esta forma del algoritmo considera un x_0 arbitrario, una distribución propuesta $x_p = x_n + \delta \cdot r$, con $r = U(-1,1)$. Además, se tiene $\alpha = 2.153$, $\beta = 1.153$ y se estimó $\delta = 3$ como un valor que aceptaba el 50 % de los valores.

2.2. Desarrollo

Primero, se define la función $W(x)$ tal que, para evitar errores, tenga un dominio en los reales. Para esto, la función W toma los valores menores que 0 y los convierte a 0. Esto no varía la distribución, pues $W(0) = 0$, pero evita errores en el algoritmo. Además, al igual que la Parte anterior, se utilizó `numba.njit` para acelerar y optimizar el tiempo de compilación.

Luego, se obtuvo una serie de 100 histogramas con distintas compilaciones de 1 algoritmo de Metrópolis para cada x_0. Se escogió cada x_0 de una distribución uniforme $U(0,2)$, de tal manera que su media de valores sea 1, el máximo de la distribución $W(x)$. Para cada x_0, se utilizó el algoritmo de Metrópolis con una semilla `numpy.random.seed` distinta, y se obtuvo un histograma con la distribución de los x_n, $n \leq 10^5$. Todos los histogramas contenían los mismos `bins` o límites.

Posteriormente, se calculó el promedio y desviación estándar de cada barra del histograma, con el fin de conseguir un histograma promedio y una desviación estándar para cada uno de los `bins`.

La Figura 4 muestra la distribución discreta obtenida por el algoritmo de Metrópolis, promediando 100 valores para cada `bin`. El error asociado es bastante pequeño, y decrece conforme se aumenta el n máximo.

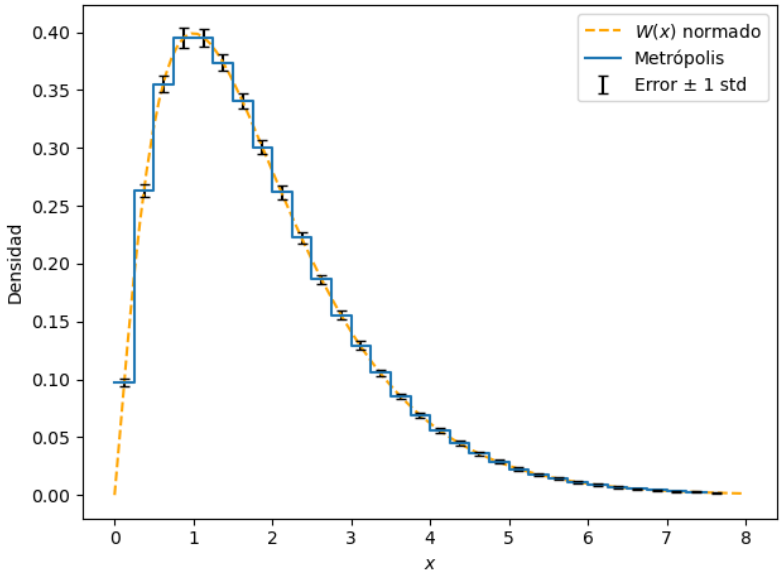

Figura 4: Histograma promedio de las variables aleatorias junto a $W(x)$ normalizado.

2.3. Discusión y Conclusiones

Finalmente, se puede concluir que el algoritmo de Metrópolis es capaz de aproximar una distribución de densidad aleatoria de forma eficaz, escogiendo puntos a partir de otra distribución propuesta. Si bien para valores pequeños de n el x_0 puede afectar la distribución final, al aumentarlo los x_n van .[en]contrando"su camino hasta los puntos más densos de la distribución de probabilidad, generando puntos de forma efectiva.

Los algoritmos de Monte Carlo o de Metrópolis son efectivos a la hora de aproximar funciones de densidad de muestreo difícil, o volúmenes multidimensionales complicados o imposibles de calcular de forma analítica.