Práctica 0: Python

Ana Martín Sánchez, Nicolás Pastore Burgos

21/09/2021

1 Descripción de la práctica

En esta práctica, se pedía un algoritmo de integración numérica, basado en el método de Monte Carlo. Este método de consiste en calcular probabilidades utilizando secuencias de números aleatorios.

Es decir, en lugar de calcular la integral de una función entre dos puntos, se busca el valor máximo M en el intervalo [f(a), f(b)], y generamos un rectángulo de anchura (b-a) y altura M. A partir de este punto, entra en juego el método de Monte Carlo; generamos puntos aleatorios dentro de ese rectángulo, y observamos si caen por debajo de la función. Repitiendo este proceso, podemos hacer una aproximación del valor de la integral.

Al utilizar el algoritmo que se pide implementar, un posible resultado sería el de la figura 1.

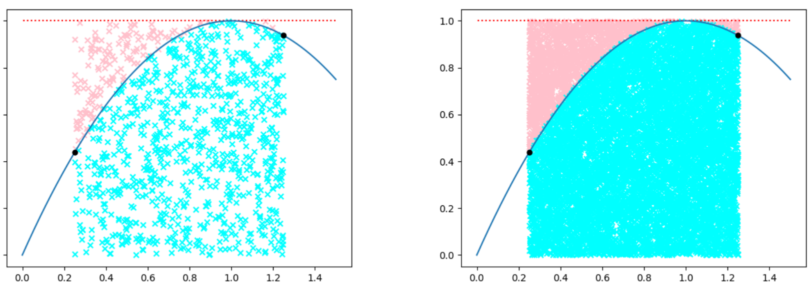

Figure 1: Después de buscar puntos aleatorios, y comprobar cuáles están por debajo de la función, podemos intuir cuál es la integral de dicha función. Cuantos más puntos calculemos, más cercana a la realidad es la aproximación que se hace.

Además, para la práctica se pide hacer dos versiones del mismo algoritmo: una iterativa, con bucles; y otra que opere sobre vecotres, con el fin de calcular los tiempos de ejecución obtenidos con ambas versiones. Es importante recordar que esta diferencia se hará más notable cuanto mayor sea el número de puntos generados.

2 Solución propuesta

2.1 Resultados obtenidos

Tras varias pruebas, se puede observar que el tiempo consumido al utilizar vectores está en el orden O(1). Es decir, se mantiene prácticamente constante, independientemente del número de puntos calculados para obtener la integral.

Sin embargo, al utilizar bucles for, el tiempo está en el orden O(n). Cuantos más puntos se crean, mayor es el recorrido del bucle; por tanto, se tarda más tiempo en recorrer todos los puntos.

En la figura 2 se ve un ejemplo de ejecución, en el que se ven los tiempos necesarios para obtener la integral. En el eje X de la gráfica, se indican el número de puntos creados; en el eje Y, el tiempo consumido por cada algoritmo, en nanosegundos. Además, como se lee en la leyenda, en azul se muestran los resultados obtenidos con el algoritmo basado en bucles y, en rojo, los obtenidos con el algoritmo basado en vectores.

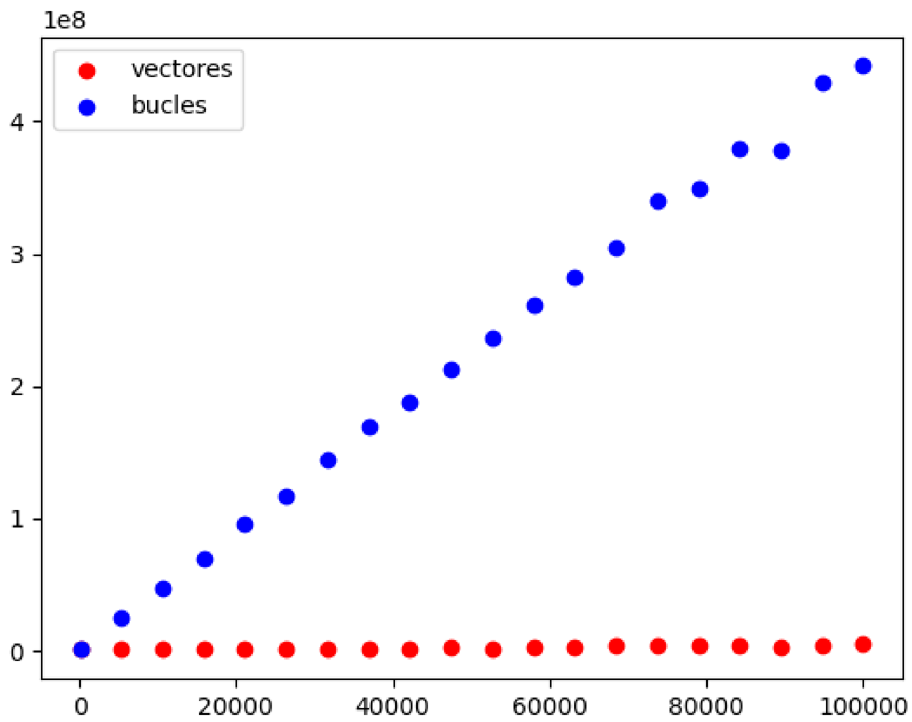

Figure 2: Se puede observar cómo, a medida que el número de puntos calculados aumenta, la diferencia de tiempos se hace mayor.

2.2 Implementación

```python
import numpy as np
import matplotlib.pyplot as plt
import time

def paint(func, a, b, maximum, random_x, random_y, random_valid, num_puntos=10000):
    # pinta la funcion en forma de una linea
    left_limit, right_limit = a - 0.25, b + 0.25
    func_x = np.linspace(left_limit, right_limit, num_puntos)
    #func_y = np.array([func(x) for x in func_x])
    # se vectoriza para que funcione con funciones constantes
    func_y = np.array(np.vectorize(func)((func_x)))
    plt.plot(func_x, func_y)

    # pinta los puntos aleatorios
    under_x, under_y = [], []
    over_x, over_y = [], []

    for i in range(num_puntos):
        if random_valid[i]:
            under_x.append(random_x[i])
            under_y.append(random_y[i])
        else:
            over_x.append(random_x[i])
            over_y.append(random_y[i])

    plt.scatter(over_x, over_y, color="pink", marker="x", s=30)
    plt.scatter(under_x, under_y, color="cyan", marker="x", s=30)

    # pinta los puntos a y b y el maximo
    plt.plot(a, func(a), marker=".", markersize=10, color="black")
    plt.plot(b, func(b), marker=".", markersize=10, color="black")
    plt.hlines(maximum, right_limit, left_limit, "r", linestyles="dotted")

    plt.show()

def integra_mc(func, a, b, num_puntos=10000):

    startTime = time.time_ns()

    # calcula el maximo de una funcion
    maximum = np.max(func(np.linspace(a, b, 1000)))

    # obtiene puntos aleatorios
    random_x = np.array(np.random.rand(num_puntos)) * (b - a) + a
    random_y = np.random.rand(num_puntos) * maximum

    # compara esos puntos con los de la funcion
    func_points_y = func(random_x)
    random_valid = np.array(random_y <= func_points_y)

```

```python
    # calcula el area
    # valid_points = np.count_nonzero(random_valid)
    # ratio = valid_points / num_puntos
    # area = ratio * (b-a) * maximum
    area = (np.count_nonzero(random_valid) / num_puntos) * (b-a) * maximum

    print("The area of the integral is aproximately:", area)

    # pinta el resultado under
    #paint(func, a, b, maximum, random_x, random_y, random_valid, num_puntos)

    return time.time_ns() - startTime

def integra_mc_for(func, a, b, num_puntos=10000):

    startTime = time.time_ns()

    maximum = max([func(i) for i in np.arange(a, b, 0.01)])

    random_x = [(np.random.rand() * (b-a) + a) for i in range(num_puntos)]

    random_y = [np.random.rand() * maximum for i in range(num_puntos)]

    func_points_y = [func(x) for x in random_x]

    random_valid = [random_y[i] <= func_points_y[i] for i in range(num_puntos)]

    area = (sum(random_valid) / num_puntos) * (b-a) * maximum
    print("The area of the integral is aproximately:", area)

    return time.time_ns() - startTime

def square(x):
    return -((x - 1) ** 2) + 1

def lineal(x):
    return 3*x + 2

def constant(x):
    return 7

def sin(x):
    return np.sin(x)

def compara_tiempos():
    sizes = np.linspace(100, 100000, 20)

    print(sizes.shape)

    times_np = []
    times_p = []
```

```python
    for size in sizes:
        a = 0.25
        b = 1.25
        times_p += [integra_mc_for(square, 0.25, 1.25, int(size))]
        times_np += [integra_mc(square, 0.25, 1.25, int(size))]

    plt.figure()
    plt.scatter(sizes, times_np, c='red', label='vectores')
    plt.scatter(sizes, times_p, c='blue', label='bucles')
    plt.legend()
    plt.show()

if __name__ == "__main__":
    #resnp = integra_mc(lineal, 0.25, 1.25)
    #resnp = integra_mc(constant, 0.25, 1.25)
    #resnp = integra_mc(sin, 0, 3.14)

    # used to calculate the time difference between the iterative method and the vector-based method
    # resnp = integra_mc(square, 0.25, 1.25)
    # print("Time of numpy operations:", resnp)
    # resp = integra_mc_for(square, 0.25, 1.25)
    # print("Time of python fors:", resp)
    # print(resnp / resp)

    compara_tiempos()
```

main.py