Práctica 0 – Python

Realizada por Mario Blanco Domínguez y Juan Tecedor Roa

- Objetivo de la práctica

El objetivo de la práctica es tomar contacto con Python y con las bibliotecas numpy y matplotlib.

Se trata de desarrollar de implementar un algoritmo de integración(de cualquier función, en nuestro caso utilizamos la función cuadrado) por el método de Monte-Carlo y comparar el tiempo de ejecución visualmente entre dos versiones: una que utiliza vectores numpy y otra que realiza la tarea iterativamente. La versión que utiliza vectores de numpy debería tener un mejor resultado.

- Código de la práctica

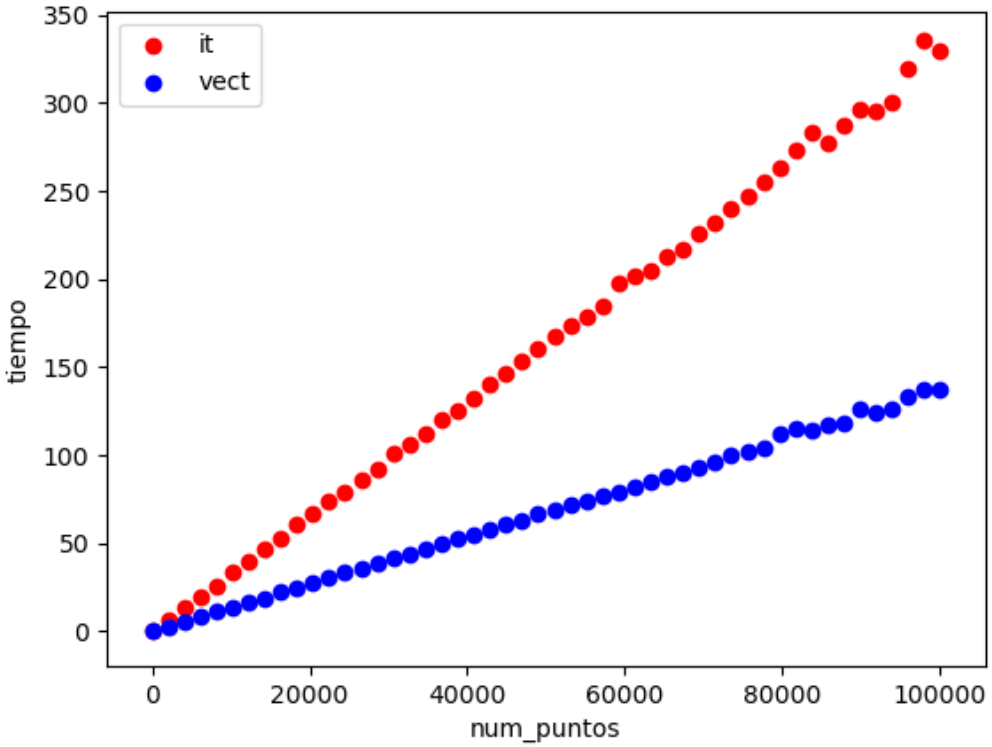
```
import numpy as np
import matplotlib.pyplot as plt
import time


def cuadrado(x):
    return x * x


def integra_mc_it(fun, a, b, num_puntos=10000):
    """Calcula la integral de fun entre a y b por Monte Carlo co
n bucles"""
    count = 0
    eje_y = fun(np.linspace(a, b, num_puntos))
    maximo_fun= max(eje_y)

    for i in range(num_puntos):
        x = np.random.uniform(a, b)
        y = np.random.uniform(0, maximo_fun)
        if y < fun(x):
            count += 1

    integral = count / num_puntos * (b - a)  * maximo_fun
    return integral


def integra_mc_vect(fun, a, b, num_puntos=10000):
    """Calcula la integral de fun entre a y b por Monte Carlo si
n bucles"""
    eje_y = fun(np.linspace(a, b, num_puntos))
    maximo_fun = max(eje_y)
```

```
    x = np.random.uniform(a, b, num_puntos)
    y = np.random.uniform(0, maximo_fun, num_puntos)
    count = sum(y < fun(x))

    integral = count / num_puntos * (b - a)  * maximo_fun
    return integral


def compara_tiempos():
    """Compara tiempos entre integra_mc_vect y integra_mc_it"""

    num_puntos = 100000
    a = 1
    b = 3

    times_it = []
    times_vect = []

    sizes = np.linspace(1, num_puntos, 50)
    for i in sizes:
        t1 = time.process_time()
        integra_mc_it(cuadrado, a, b, int(i))
        t2 = time.process_time()
        elapsed_time = 1000 * (t2 - t1)
        times_it += [elapsed_time]

        t1 = time.process_time()
        integra_mc_vect(cuadrado, a, b, int(i))
        t2 = time.process_time()
        elapsed_time = 1000 * (t2 - t1)
        times_vect += [elapsed_time]

    plt.figure()
    plt.scatter(sizes, times_it, c='red', label='it')
    plt.scatter(sizes, times_vect, c='blue', label='vect')
    plt.legend()
    plt.xlabel('num_puntos')
    plt.ylabel('tiempo')
    plt.savefig('./time.png')
    plt.close()

compara_tiempos()
```

- Resultados de ejecución

En esta gráfica se aprecia cómo el algoritmo iterativo, según se van generando más puntos aleatorios, es más lento que el algoritmo que utiliza vectores de numpy.

Para la prueba se ha ejecutado el algoritmo probando de 1 a 100000 puntos y se han realizado 50 pruebas por algoritmo.



- Conclusiones

Se concluye que el uso de vectores numpy es muy útil para tareas con arrays, por lo que en tareas que se utilicen arrays con muchos datos será conveniente utilizarlos en vez de los nativos de Python.