Práctica 0: Python

David Godoy Ruiz
Eva Lucas Leiro

Calcular el máximo:

Tanto el método iterativo como el método con vectores de numpy calculan el máximo aproximado de la función de la misma forma: cogiendo muchos puntos entre el punto a y el punto b, calculando sus imágenes(y) y obteniendo la mayor de todas. Esto no garantiza que sea el punto mayor de la función, pero es una aproximación suficiente para ambos cálculos.

Método Iterativo:

```
def integra_mc(fun, a, b, num_puntos = 10000):
    dentro = 0
    max = np.amax(fun(np.linspace(a,b,num_puntos)))

    tic = time.process_time()

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

    toc = time.process_time()
    sol = (dentro/num_puntos) * (b-a) * max
    return 1000 * (toc-tic)
```

El método iterativo consiste en utilizar un bucle que realiza iteraciones iguales al número de puntos pedidos por parámetro. En cada iteración se saca una x e y aleatorias, y esta última se compara con la imagen de dicha x, llevando la cuenta de los valores menores, es decir, que se cuentan los puntos dentro del área de la función (la integral). A partir de ahí se puede calcular la integral de la función a partir de la fórmula dada:

$$I \approx \frac{N_{debajo}}{N_{total}}(b-a)M$$

Método con vectores:

```
def integra_mc_vec(fun, a, b, num_puntos = 10000):
    max = np.amax(fun(np.linspace(a,b,num_puntos)))

    tic = time.process_time()

    x = np.random.uniform(a, b, num_puntos)
    y = fun(x)
    yrand = np.random.uniform(0, max, num_puntos)

    toc = time.process_time()
    sol = (sum(yrand<y)/num_puntos) * (b-a) * max
    return 1000 * (toc-tic)
```

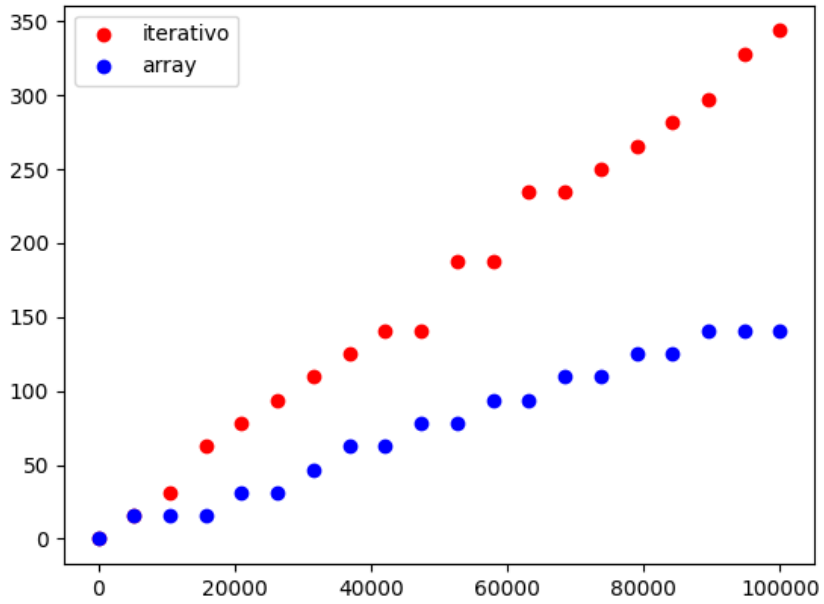
El método con vectores de numpy se realiza de forma similar, solo que para generar los puntos x e y aleatorios, y las imágenes de dichas x se utilizan vectores de numpy generados aleatoriamente dentro de los intervalos de la función.

A la hora de hacer el cálculo de la integral basta con comparar en una sola instrucción los puntos dentro del área de la función (o sea, los que están dentro de la integral) contra todos los puntos creados de la misma forma que en el método anterior.

Análisis de tiempo:

Al calcular el tiempo de ejecución como se mencionaba en el enunciado, se ve claramente que el método con vectores es mucho más rápido, y cuantos más puntos deben dibujarse, más se nota la diferencia entre los tiempos, donde el método iterativo crece de forma más o menos lineal mientras que el vectorial crece con menor pendiente, probablemente de forma logarítmica

Gráfica comparativa de tiempos de los algoritmos



```
def compara_tiempos():
    sizes = np.linspace(10, 100000, 20, dtype=int)
    times_it = []
    times_vec = []
    for size in sizes:
        times_it += [integra_mc(funcion, 1, 10, size)]
        times_vec += [integra_mc_vec(funcion, 1, 10, size)]

    plt.figure()
    plt.scatter(sizes, times_it, c='red', label='iterativo')
    plt.scatter(sizes, times_vec, c='blue', label='array')
    plt.legend()
    plt.savefig('time.png')
```

```
def funcion(x):
    return 2*x
```

La comparación de tiempos ha sido realizada con el mismo método mostrado en la ayuda, en este caso con 20 puntos entre 10 y 100000 para la integral de 2x en el intervalo [1-10]