



Universidad Complutense de Madrid
Facultad de Informática
Aprendizaje Automático y Big Data



Memoria Práctica 0.

Profesor:

- Alberto Díaz Esteban.

Alumnos:

- Marina de la Cruz López.

- Diego Alejandro Rodríguez Pereira.

Código

```
import time
import numpy as np
import random as rn
import scipy.integrate
import math
import matplotlib.pyplot as plt
```

```
def integra_mc(fun, a, b, num_puntos=10000):
    lista_puntos = list()
    MAX = float('-inf')
    MIN = float('inf')
    for i in range(0, num_puntos):
        temp = fun(rn.uniform(a,b))
        if(temp > MAX):
            MAX = temp
        if(temp < MIN):
            MIN = temp

    for i in range(0, num_puntos):
        #X                                     #Y
        lista_puntos.append( (rn.uniform(a, b), rn.uniform(0, MAX)) )

    num_debajo = 0
    for punto in lista_puntos:
        if(punto[1] < fun(punto[0])):
            num_debajo = num_debajo+1

    return (num_debajo/num_puntos)*(b-a)*MAX
```

```
def integra_mc_vec(fun, a, b, num_puntos):

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

    puntosY = np.array([fun(xi) for xi in x])

    num_debajo = np.sum(y < puntosY)

    res = num_debajo/num_puntos*(b-a)*MAX

    return res
```

```
def compara_tiempos():

    sizes = np.linspace(100, 10000000, 20)

    times_iter = []
    times_vec = []
```

```

for size in sizes:

    a = 1
    b = 100

    tic = time.process_time()
    res_iter = integra_mc(math.sin, a, b, int(size))
    toc = time.process_time()
    times_iter += [1000 * (toc - tic)]

    tic = time.process_time()
    res_vec = integra_mc_vec(math.sin, a, b, int(size))
    toc = time.process_time()
    times_vec += [1000 * (toc - tic)]

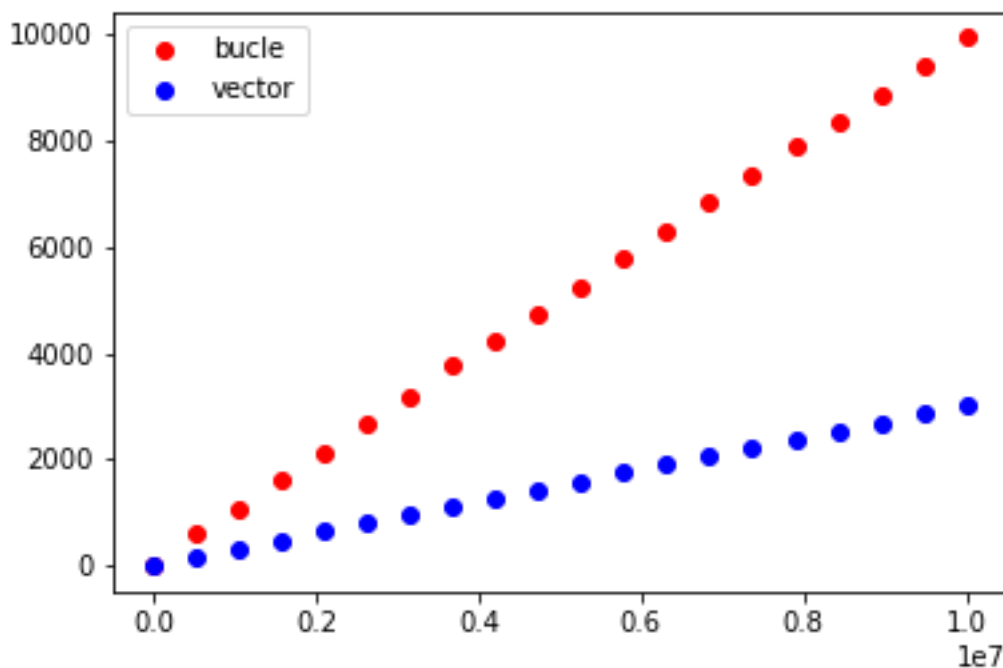
    p1 = len(sizes)
    p2 = len(times_iter)

    #print("size1", p1, "size2", p2)

plt.figure()
plt.scatter(sizes, times_iter, c='red', label='bucle')
plt.scatter(sizes, times_vec, c='blue', label='vector')
plt.legend()
plt.savefig('compara_tiempos_dot.png')

```

```
compara_tiempos()
```



Conclusiones

La biblioteca de Numpy hace que trabajar con las estructuras de arrays sea más fácil y sencillo. Por lo que además de obtener un código más compacto y legible proporciona una mayor velocidad de ejecución que al utilizar los arrays propios de Python.

Esto se puede observar en la gráfica con los tiempos obtenidos de ambos códigos, siendo los puntos rojos los obtenidos con los arrays convencionales, y los puntos azules los tiempos obtenidos utilizando los arrays de Numpy.