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()

Chart, scatter chart

Description automatically generated

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.