Práctica 0: Python

Manuel Hernández Nájera-Alesón
Mario Jiménez Contreras

Para abordar la práctica comenzamos generando la gráfica de la función

$$x^2$$

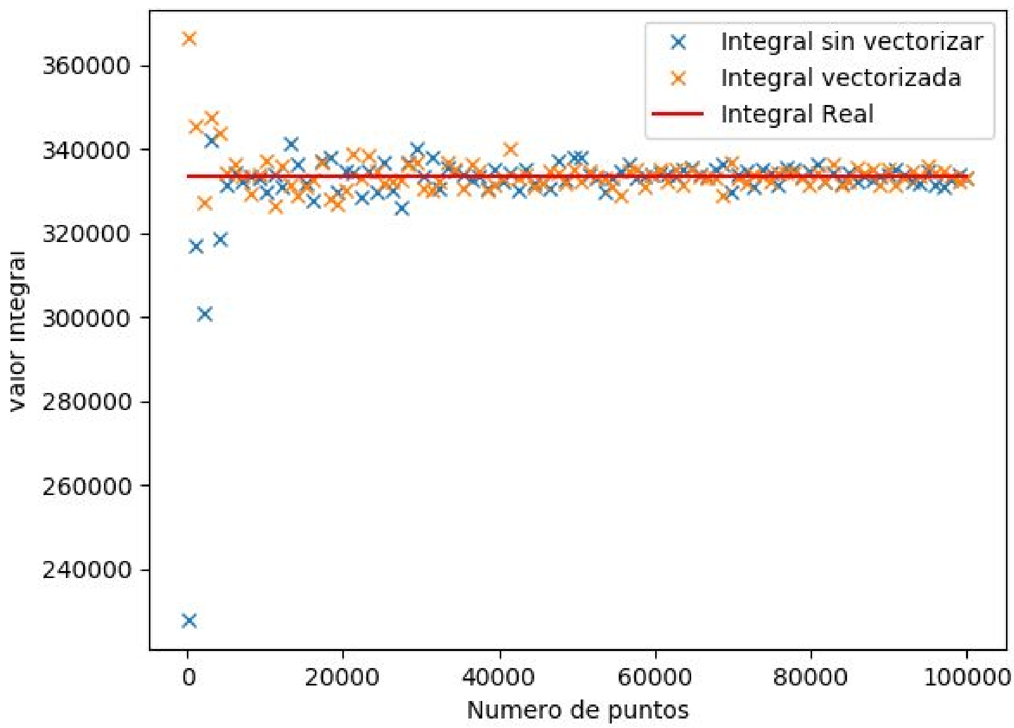
mediante la creación de dos arrays, uno para guardar los valores de x y otro para guardar los distintos valores de f(x), y pasar a matplotlib estos dos arrays para que llevase a cabo el dibujado.

Después generamos otros dos arrays, con coordenadas x e y que definen puntos en el marco de la gráfica para, posteriormente, contar cuántos quedan por debajo de esta y hallar así el área bajo la curva.

Para hallar el área tenemos dos funciones:

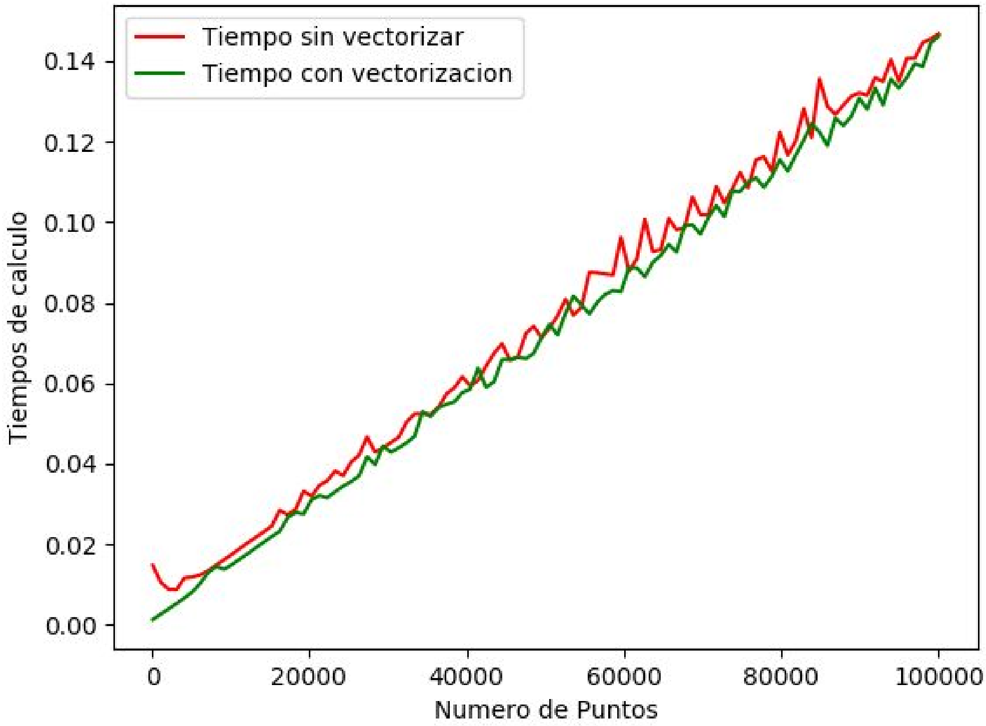
- integra_mc: usa un for convencional, comparando f(x) con el valor de los puntos generados y sumando el número de puntos por debajo de f(x)
- integra_mc_vector: utilizando vectorización y funciones que facilitan el recorrido de arrays realizamos la misma acción que con el for convencional. En concreto son las funciones max y sum las que más tiempo y líneas nos ahorran.

Dando como resultado las siguientes gráficas:



Representación de la integral

Y con diferentes tiempos, apreciables en esta imagen:



El código utilizado es el siguiente:

```
# coding=utf-8 #Para que no de fallos de caracteres no-ASCII
# Practica 0 de AAMD
# Implementación en python de la integración por el metodo
# de Monte Carlo
# Manuel Hernandez y Mario Jimenez

import random as rnd
import matplotlib as mp
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import scipy.integrate as integ
import time

# Integral con for convencional
def integra_mc (fun, a, b, num_puntos = 10000):
    under = 0.0 # Num de puntos bajo la curva

    maxy = p_y[0]
    miny = p_y[0]
    # Hallamos los valores máximo y mínimo que alcanza la y para
    # la posterior generación de puntos aleatorios
```

```
    for i in range (0, len(p_y)):
        if p_y[i] < miny:
            miny = p_y[i]
        elif p_y[i] > maxy:
            maxy = p_y[i]

    # Creamos puntos de manera aleatoria dentro del marco de la grafica
    xx = np.array([rnd.uniform(a, float(b)) for n in range(0,
num_puntos)])
    yy = np.array([rnd.uniform(miny, maxy) for n in range(0,
num_puntos)])

    # Sumamos todos los puntos por debajo de la función
    i = 0
    for x in xx:
        if(yy[i] < fun(xx[i])):
            under = under +1
        i = i+1

    return float(under / num_puntos)*(b-a)*maxy

# Integral optimizada mediante vectorización y operaciones de numpy
def integra_mc_vector(fun, a, b, num_puntos = 10000):
    under = 0.0

    maxy = max(p_y)
    miny = min(p_y)

    xx = np.random.uniform (a, b, num_puntos)
    yy = np.random.uniform (miny, maxy, num_puntos)
    aux = yy < fun(xx)

    under = sum (aux)

    return float(under / num_puntos)*(b-a)*maxy

#Función para calcular el tiempo de ejecucion de func
def calctiempo(func,integr, nump):
    tic = time.process_time()
    temp =  func(foo, a, b, int(nump))
    toc = time.process_time()
    integr.append(temp)
    return toc - tic

#funcion a integrar
def foo(x):
    return x**2

print("Por favor introduzca el intervalo a integrar A B")
inp = input()
a = int(inp.split(' ')[0])
b = int(inp.split(' ')[1])
fun = foo #Funcion a integrar
```

```
p_x = []

# Llenamos una lista con 10 mil puntos aleatorios en el segmento a, b
# y otra lista de coordenadas y calculada con el valor de los distintos
f(x)
p_x = np.linspace(a,b, 10000)
p_y = np.array([fun(n) for n in p_x ])

# Arrays para la visualización de rendimiento de cada algoritmo
numpunts=np.linspace(100, 100000, 100)
inte = []
intevec = []

realinte = integ.quad(foo, a, b)

valtemps = np.array([calctiempo (integra_mc, inte, x) for x in
numpunts])
valtempsvec = np.array([calctiempo (integra_mc_vector, intevec, x) for
x in numpunts])

print('Integral de Numpy: ', realinte)
print('Integral con for: ', integra_mc(fun, a, b, 10000))
print('Integral con vectorización: ', integra_mc_vector(fun, a, b,
10000))

#Pintado de resultados
plt.figure()

# Grafica del rendimiento
plt.plot(numpunts, valtemps, '-', label = "Tiempo sin vectorizar",
color = "red")
plt.plot(numpunts, valtempsvec, '-', label = "Tiempo con
vectorizacion", color = "green")
plt.xlabel("Numero de Puntos")
plt.ylabel("Tiempos de calculo")
plt.legend()

# Grafica con las integrales
plt.figure()
plt.plot(numpunts, inte, 'x', label = "Integral sin vectorizar")
plt.plot(numpunts,intevec, 'x', label = "Integral vectorizada")
plt.plot(numpunts, [realinte[0] for x in numpunts],'-',  color = "red",
label = "Integral Real")
plt.xlabel("Numero de puntos")
plt.ylabel("Valor Integral")
plt.legend()

plt.show()
```