Aprendizaje Automático y Big Data, Práctica 0

Romain Contrain y Guillermo Martín Sánchez

Febrero 2020

1 Objetivo

El objetivo de esta práctica es implementar un algoritmo que calcule una aproximación a la integral de una función f por el método de Monte Carlo. Esto es, aproxime $\int_a^b f(x)dx \approx \frac{N_{debajo}}{N_{total}}(b-a)M$ donde $M = \max\limits_{[a,b]}(f)$.

En concreto hemos calculado la integral de la función $f(x) = x^2$ en $[0,3]$ y comparado el valor estimado con el valor real. Además, lo hemos hecho de dos formas: usando la librería *numpy* y no usando la librería, con la finalidad de medir los tiempos de diferencia para diferente N_{total}

2 Resultados

Como podemos ver en la Figura 1, ambas implementaciones dan resultados similares, que están bastante cercanos al resultado exacto ($\int_0^3 x^2dx = 9$) y se acercan más y más cuánto mayor sea el número de puntos N_{total}.

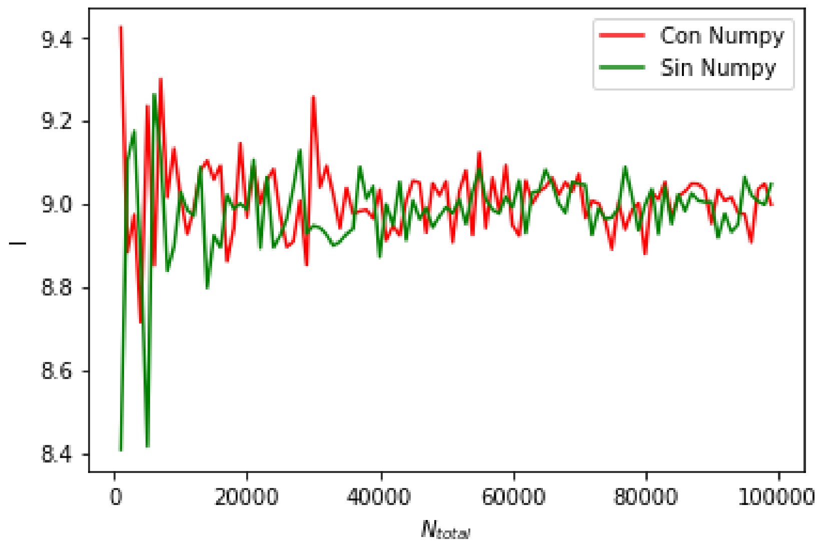

Figure 1: Valor de la integral calculado para diferente número de puntos N_{total}

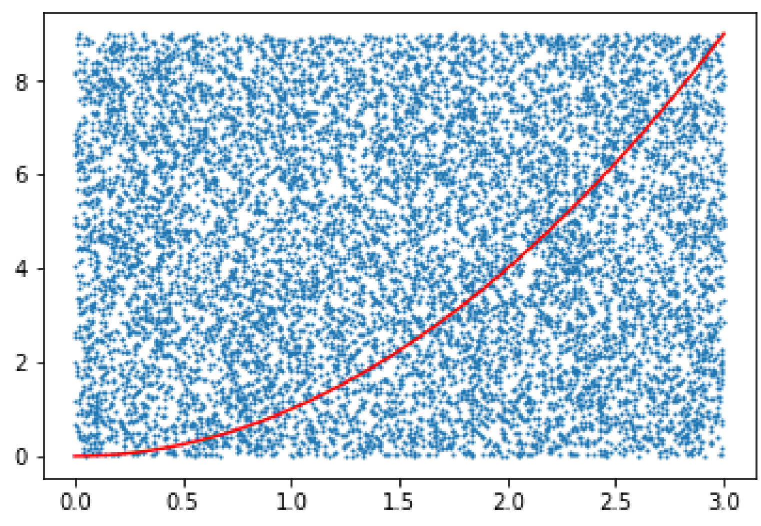

Figure 2: El área debajo de la curva x^2, explorada con $N_{total} = 10000$ puntos

Por otro lado, respecto a los tiempos, observamos en la Figura 3 que la implementación usando *numpy* es mucho más rápida (dando tiempos considerados $0ms$ por el propio programa) y de coste casi constante (no parece crecer conforme aumentamos el número de puntos). Sin embargo, la implementación sin usar *numpy* sí crece significativamente de una forma que parece lineal con respecto al número de puntos.

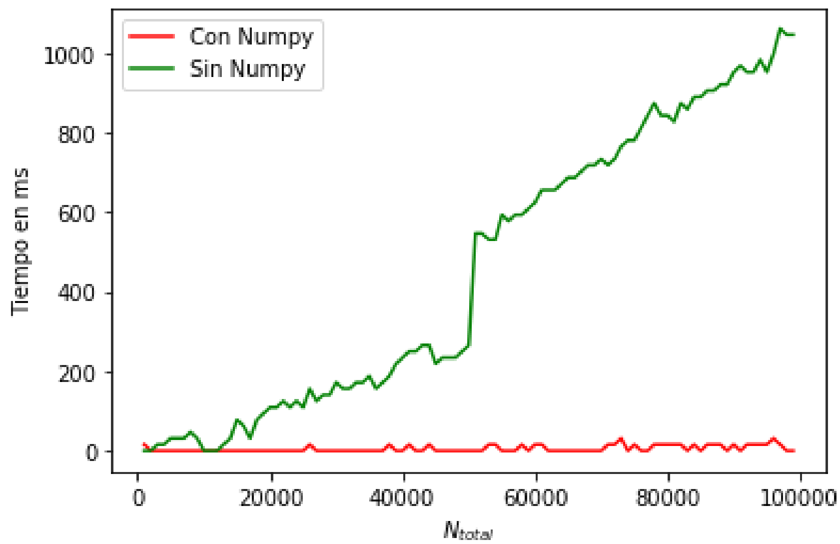

Figure 3: Tiempo en ms que tarda cada una de las implementaciones en estimar la integral con diferente número de puntos

3 Conclusiones

Concluimos que la aproximación de la integral mediante el método de Monte Carlo es bastante buena cuando N_{total} es muy grande. Por otro lado, concluimos que explotar las herramientas que *numpy* nos proporciona no sólo hace el código más rápido de programar sino también mucho más rápido de ejecutar.

4 Código

```
# -*- coding: utf-8 -*-
"""
Practica 0
"""

import numpy as np
import time
```

```python
import matplotlib.pyplot as plt

# Usando numpy
def integra_mc(f,a,b,num_puntos=10000):
    x = np.linspace(a,b,num_puntos)
    y = f(x)
    M = np.amax(y)
    p = np.random.rand(num_puntos,2)
    p[:,0] = p[:,0]*(b-a)+a
    p[:,1] = p[:,1]*M
    r = p[:,1] < f(p[:,0])
    N = np.sum(r)
    I = (N/num_puntos) * (b-a) * M
    return I,p

# Sin usar numpy
def integra_mc2(f,a,b,num_puntos=10000):
    x = a
    step = (b-a)/num_puntos
    M = 0
    for i in range(num_puntos):
        M = max(M,f(x))
        x = x + step
    p = np.random.rand(num_puntos,2)
    N = 0
    for j in range(num_puntos):
        p[j,0] = p[j,0]*(b-a)+a
        p[j,1] = p[j,1]*M
        N = N + (p[j,1] < f(p[j,0]))
    I = (N/num_puntos) * (b-a) * M
    return I,p

f = lambda x: x**2
a = 0
b = 3
tic = time.process_time()
I,p = integra_mc(f,a,b,100000)
toc = time.process_time()
print("Integral calculada:",I, "en tiempo",(toc-tic)*1000, " ms")

tic = time.process_time()
I,p = integra_mc2(f,a,b,100000)
toc = time.process_time()
print("Integral calculada:",I, "en tiempo",(toc-tic)*1000," ms")

x = np.linspace(a,b,1000)
```

```
plt.scatter(p[:,0],p[:,1], s=1)
plt.plot(x,f(x), 'r')

#%%
"""
Medir tiempos y exactitud
"""

tnp = []
t = []
Inp = []
Isnp = []
num_puntos = range(1000,100000,1000)
for n in num_puntos:
    tic = time.process_time()
    I,p = integra_mc(f,a,b,n)
    toc = time.process_time()
    tnp.append((toc-tic)*1000)
    Inp.append(I)

    tic = time.process_time()
    I,p = integra_mc2(f,a,b,n)
    toc = time.process_time()
    t.append((toc-tic)*1000)
    Isnp.append(I)

plt1, = plt.plot(num_puntos,tnp, 'r', label = 'Con Numpy')
plt2, = plt.plot(num_puntos,t, 'g', label = 'Sin Numpy')
plt.legend(handles=[plt1,plt2])
plt.xlabel('$N_{total}$')
plt.ylabel('Tiempo en ms')
plt.show()

plt1, = plt.plot(num_puntos,Inp, 'r', label = 'Con Numpy')
plt2, = plt.plot(num_puntos,Isnp, 'g', label = 'Sin Numpy')
plt.legend(handles=[plt1,plt2])
plt.xlabel('$N_{total}$')
plt.ylabel('I')
plt.show()
```