PRÁCTICA 0

Integración con el método de Monte Carlo en Python

Autoras: María García Raldúa, Luisa

Para la práctica hemos usado la función ejemplo con límites de 0 a pi con la función seno.

Hemos usado dos métodos, hallar el área con vectores y con iteraciones simples.

Monte Carlo con vectores

Hemos usado la función de linspace de numpy para generar el espacio de la función en los límites con un número de puntos y hemos obtenido el máximo de la función con “max”.

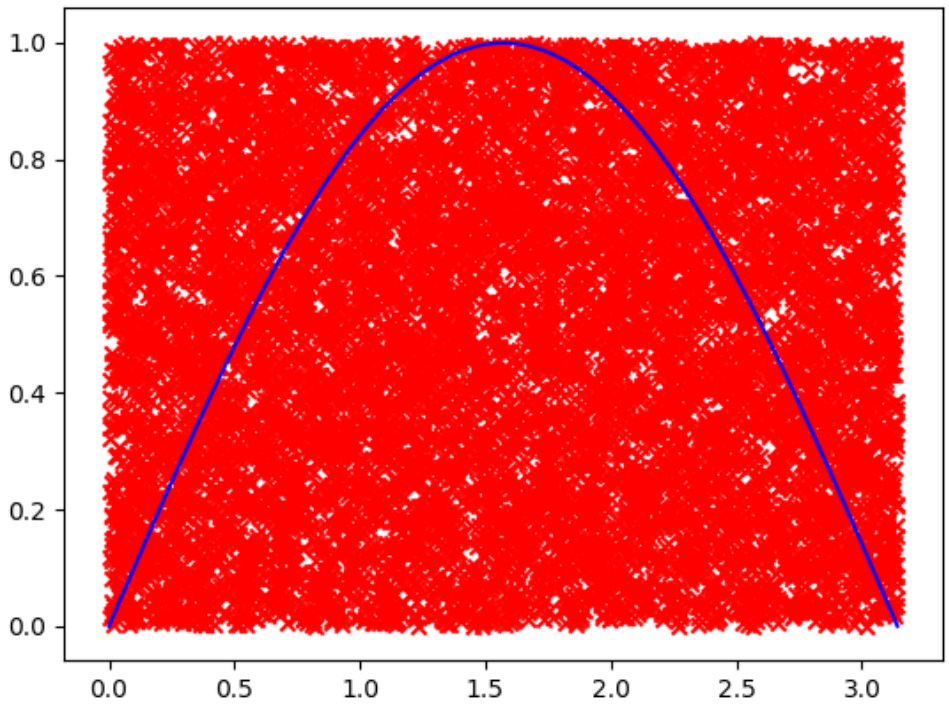
Obtenemos num_puntos aleatorios entre 0 y el máximo con la función random de numpy y aplicamos la fórmula cogiendo los puntos que quedan por debajo de la función.

A su vez calculamos el tiempo con time.process_time().

El resultado del método siempre es aproximado o igual a 2.0

Tiempo: 15.625

Gráfica:



Monte Carlo con iteraciones

Para simular el comportamiento de np.linspace, usamos un array y un incremento de (b - a)/num_puntos y mediante un bucle rellenamos el array.

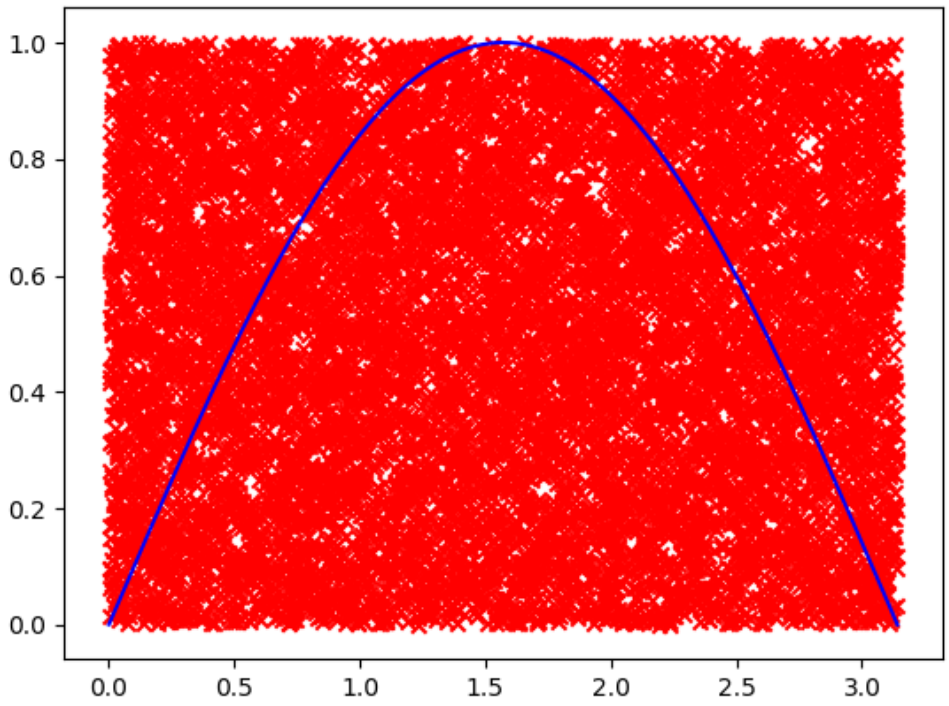
Luego generamos otro array de y con puntos random en el espacio entre 0 y el máximo (obtenido de aplicar max en el array de x).

Con un bucle for vamos contando los puntos que hay por debajo de la función y con eso hacemos la fórmula de Monte Carlo.

Resultado igual o aproximado a 2.0.

Tiempo: 31.25

Gráfica:




```
import numpy as np
from matplotlib import pyplot as plt
from scipy import integrate as integrate
import random as rnd
import time

# Integración con vectores
def integrate_vector(fun,a,b,num_puntos=10000):
```

```
    # Capturamos el tiempo inicial
    tic = time.process_time()

    # Creamos el espacio
    x = np.linspace(a,b,num_puntos)

    y1 = 0
    # Máximo de la función de x
    y2 = max(fun(x))

    # Generamos puntos random en ese espacio
    y_r = np.random.uniform(y1,y2,num_puntos)

    # Aplicamos la fórmula con los puntos que están por
    # debajo de la función
    numtotal = np.sum(fun(x) > y_r)
    res = (numtotal)*(b-a)*y2 / num_puntos

    resultado = integrate.quad(fun,a,b)

    # Capturamos el tiempo final y calculamos el total
    toc = time.process_time()
    temp = 1000 * (toc - tic)

    print(temp)
    print(res)
    # Creamos la figura
    plt.figure()
    plt.plot(x, fun(x), color="blue")
    plt.scatter(x,y_r,color="red", marker='x')
    plt.show()
    print(resultado)
    return temp

# Integración con bucles iterativos
def integrate_iter(fun,a,b,num_puntos=10000):
    tic = time.process_time()
    randx = []
    cont = 0
    # Incremento que vamos a usar para ir avanzando
    # en el espacio
    incr = (b - a)/num_puntos
    i = a

    # Rellenamos el espacio en x y calculamos el máximo
    while(i < b):
        x = i + incr
        randx.append(x)
        i = x
```

```
    maximo = max(fun(randx))

    # Generamos el array de y con los puntos aleatorios
    # en el rango num_puntos
    y = np.array([rnd.uniform(0,maximo)for n in range(0,num_puntos)])

    # Contamos los puntos que están por debajo de la función
    for c in range(len(randx)):
        if(y[c] < fun(randx[c])):
            cont = cont + 1

    res = float(cont / num_puntos)*(b - a)*maximo

    toc = time.process_time()
    temp = 1000 * (toc - tic)

    print(temp)
    print(res)

    plt.figure()
    plt.plot(randx, fun(randx), color="blue")
    plt.scatter(randx,y,color="red", marker='x')
    plt.show()

    return temp

def integra_mc(fun, a ,b ,num_puntos=10000):
    temp_vec = integrate_vector(fun,a,b)
    temp_iter = integrate_iter(fun,a,b)
    x = np.arange(2)
    plt.figure()
    tiempos = [temp_vec,temp_iter]
    plt.bar(x,tiempos,width = 0.5)
    plt.xticks(x,('Tiempo vectores','Tiempo Iterador'))
    plt.show()

integra_mc(np.sin,0,np.pi)
```

Cálculo de tiempo

Por último hacemos la comparación del tiempo cogiendo la duración de cada método antes de las funciones para mostrarlas (es decir, antes de usar matplotlib) y hacemos una gráfica de barras en la que se observa que con vectores se tarda menos en aplicar el método.