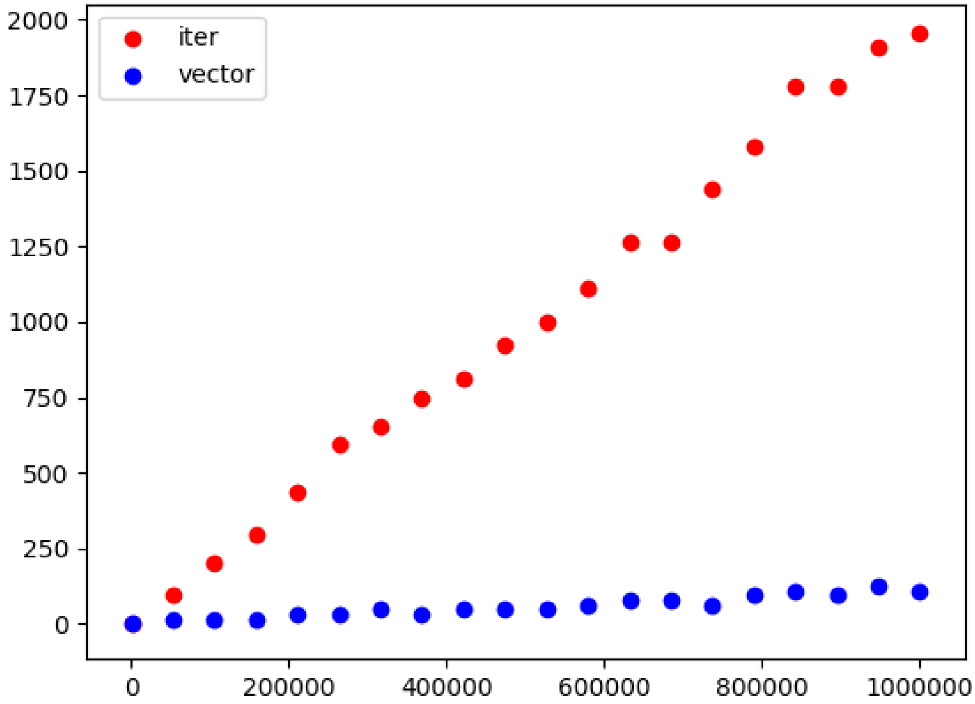
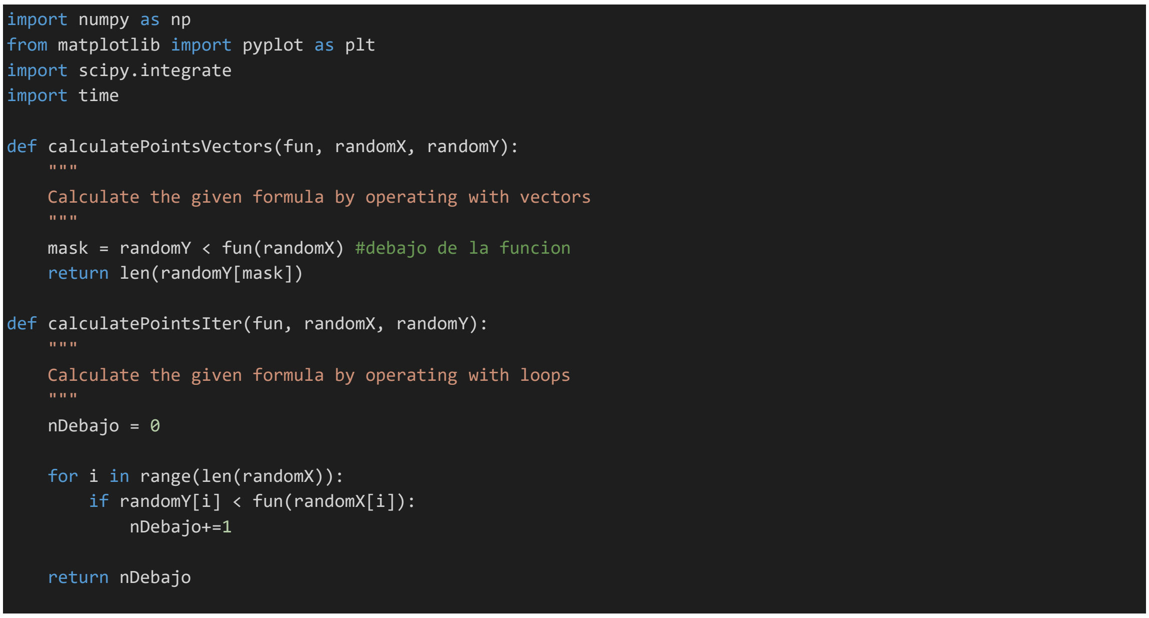


Gráfica obtenida a partir del código que se muestra a continuación.



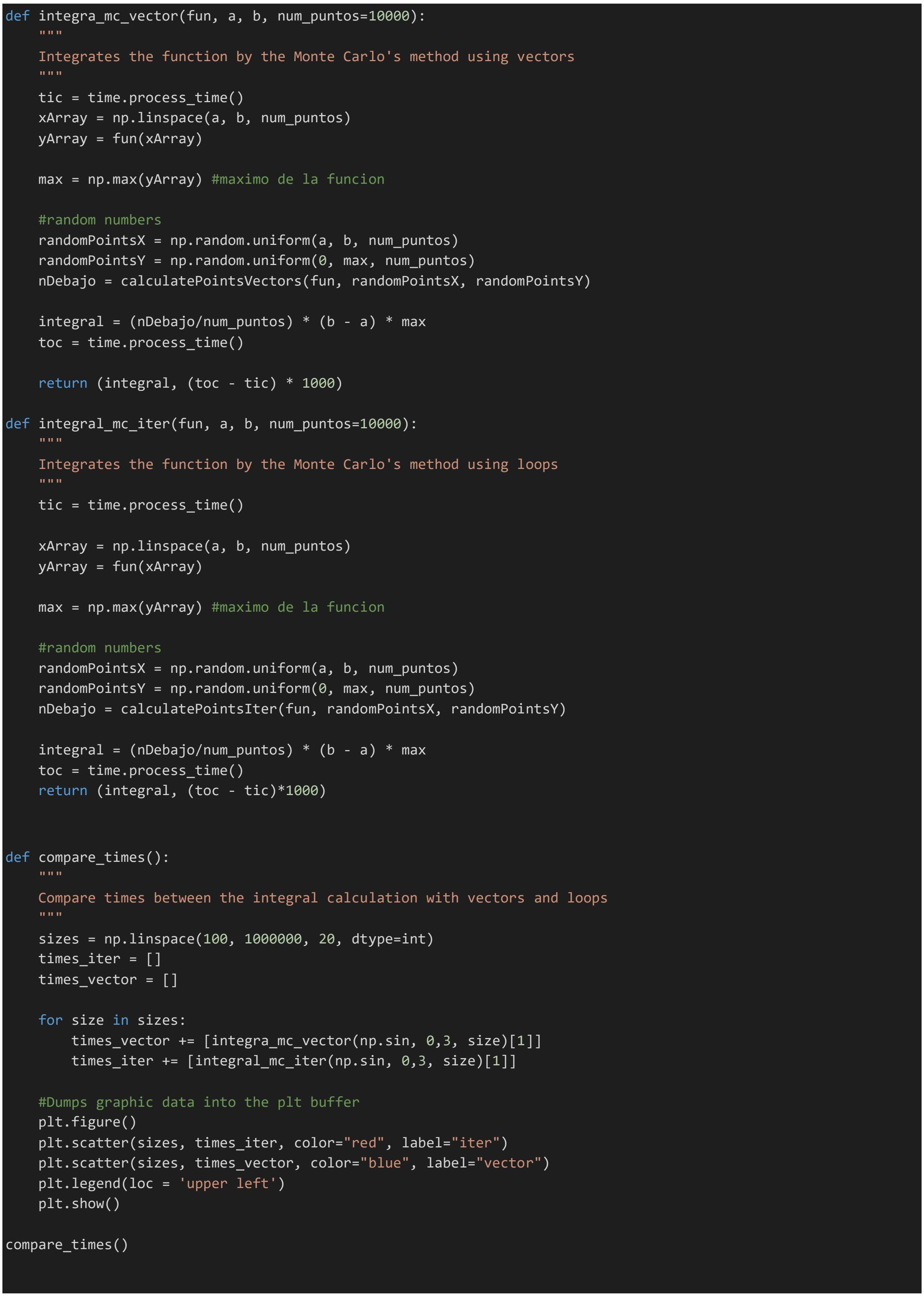
```
import numpy as np
from matplotlib import pyplot as plt
import scipy.integrate
import time

def calculatePointsVectors(fun, randomX, randomY):
    """
    Calculate the given formula by operating with vectors
    """
    mask = randomY < fun(randomX) #debajo de la funcion
    return len(randomY[mask])

def calculatePointsIter(fun, randomX, randomY):
    """
    Calculate the given formula by operating with loops
    """
    nDebajo = 0

    for i in range(len(randomX)):
        if randomY[i] < fun(randomX[i]):
            nDebajo+=1

    return nDebajo
```



```
def integra_mc_vector(fun, a, b, num_puntos=10000):
    """
    Integrates the function by the Monte Carlo's method using vectors
    """
    tic = time.process_time()
    xArray = np.linspace(a, b, num_puntos)
    yArray = fun(xArray)

    max = np.max(yArray) #maximo de la funcion

    #random numbers
    randomPointsX = np.random.uniform(a, b, num_puntos)
    randomPointsY = np.random.uniform(0, max, num_puntos)
    nDebajo = calculatePointsVectors(fun, randomPointsX, randomPointsY)

    integral = (nDebajo/num_puntos) * (b - a) * max
    toc = time.process_time()

    return (integral, (toc - tic) * 1000)

def integral_mc_iter(fun, a, b, num_puntos=10000):
    """
    Integrates the function by the Monte Carlo's method using loops
    """
    tic = time.process_time()

    xArray = np.linspace(a, b, num_puntos)
    yArray = fun(xArray)

    max = np.max(yArray) #maximo de la funcion

    #random numbers
    randomPointsX = np.random.uniform(a, b, num_puntos)
    randomPointsY = np.random.uniform(0, max, num_puntos)
    nDebajo = calculatePointsIter(fun, randomPointsX, randomPointsY)

    integral = (nDebajo/num_puntos) * (b - a) * max
    toc = time.process_time()
    return (integral, (toc - tic)*1000)


def compare_times():
    """
    Compare times between the integral calculation with vectors and loops
    """
    sizes = np.linspace(100, 1000000, 20, dtype=int)
    times_iter = []
    times_vector = []

    for size in sizes:
        times_vector += [integra_mc_vector(np.sin, 0,3, size)[1]]
        times_iter += [integral_mc_iter(np.sin, 0,3, size)[1]]

    #Dumps graphic data into the plt buffer
    plt.figure()
    plt.scatter(sizes, times_iter, color="red", label="iter")
    plt.scatter(sizes, times_vector, color="blue", label="vector")
    plt.legend(loc = 'upper left')
    plt.show()

compare_times()
```