Numerical integration calculation

We have used Monte Carlo's method for the calculation of numerical integral. In this approach, for a given function f(x) in the interval of two points, a and b on the X-axis, and a maximum M on the Y-axis, we can find an approximate answer to the function's integration.

The result is obtained by calculating the probability of situating a randomly generated point in a square, below the function's curve.

Using the previously described method, we prepared two ways to simulate integral calculation, normal (non-optimized), and optimized.

First, we calculate the maximum by choosing random points placed on the f(x) curve, and we compare them. Then we position arbitrary points inside of the imaginary square between the given two points: a and b, and the maximum M. Finally, we compare if the points inside of the square are below the function.

Our code:

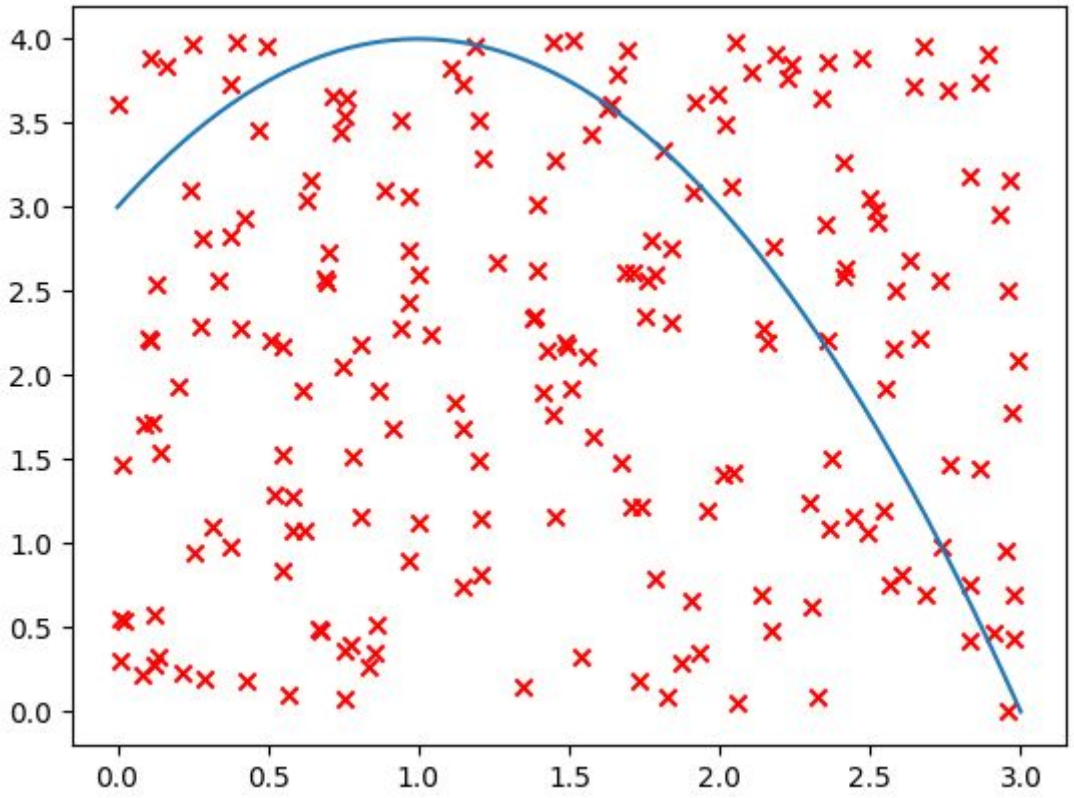
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate
import time

def fun(x):
   return -1 * x ** 2 + 2 * x + 3

def integra_mc(fun, a, b, num_puntos=10000):
   X = np.random.rand(num_puntos) * (b - a) + a
   M = max(fun(X))
   Y = np.random.rand(num_puntos) * M


   line = np.linspace(a, b, num_puntos)
   plt.scatter(X[:200], Y[:200], c = 'red', marker = 'x')
   plt.plot(line, fun(line))
   sum_nonVect(X, Y)
   N_debajo = sum_Vectorized(X, Y)

   plt.show()
   return (N_debajo / num_puntos) * (b - a) * M
```

```
def sum_nonVect(X, Y):
    tic = time.process_time()
    N_debajo = 0
    points = fun(X)
    for i in range(len(Y)):
        if(Y[i] < points[i]):
            N_debajo += 1
    toc = time.process_time()
    print("Non-vectorized sum: " + str(N_debajo) + "\n ----- Computation
time = " + str(1000*(toc - tic)) + "ms")

#difference with 1000000, with less it's 0ms
def sum_Vectorized(X, Y):
    tic = time.process_time()
    N_debajo = np.sum(Y < fun(X))
    toc = time.process_time()
    print("Vectorized sum: " + str(N_debajo) + "\n ----- Computation
time = " + str(1000*(toc - tic)) + "ms")
    return N_debajo



our_res = integra_mc(fun, 0, 3, 1000000)
true_res = scipy.integrate.quad(fun, 0, 3)

print("Result obtained by us: " + str(our_res))
print("Result obtained by the scipy.integrate.quad: " +
str(true_res[0]))
```

The results obtained executing the above code are:

```
Non-vectorized sum: 750647
----- Computation time = 249.16015999999985ms
Vectorized sum: 750647
----- Computation time = 7.638097999999927ms
Result obtained by us: 9.007763999999321
Result obtained by the scipy.integrate.quad: 9.0
```

We can see that the vectorized sum calculation time is, as expected, several times lower compared to the non-vectorized. Also, the obtained result for the integral itself is the same verifying it with the scipy function.



In the chart above we can see the curve of our function in the given interval [0,3]. The curve represents the borderline between the randomly generated points inside the interval and outside of it.

Conclusions:

Calculating the integral of a given function using the method of Monte Carlo both in an optimized and non-optimized way gave us similar results to the ones from the scipy library. As expected, the optimized version is orders of magnitude faster than the non-optimized.

Gasan Nazer and Veronika Yankova