Práctica 0: Python

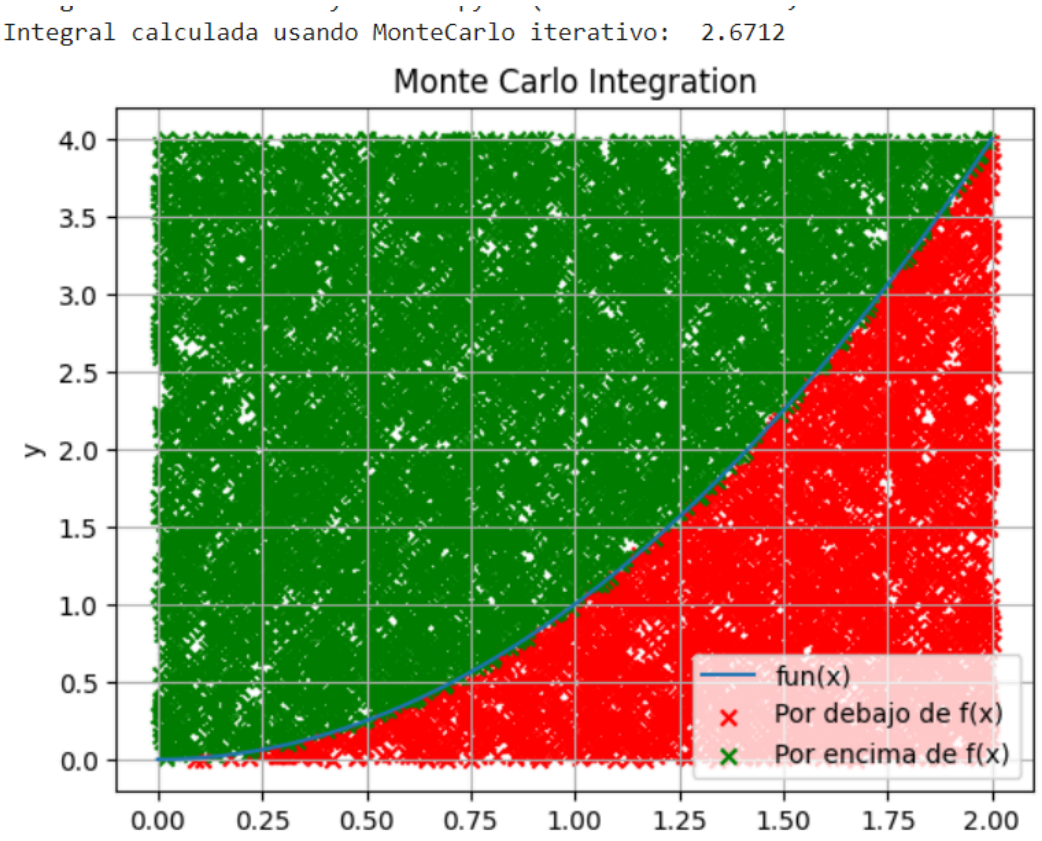
Esther Babon Arcauz

El primer paso es que el usuarios nos de un intervalo, a partir de ahí, se define el espacio en el que pueden existir los puntos aleatorios que genero a continuación. Teniendo los puntos, cuento cuantos están por debajo de la función y calculo la integral usando el método de Monte Carlo.

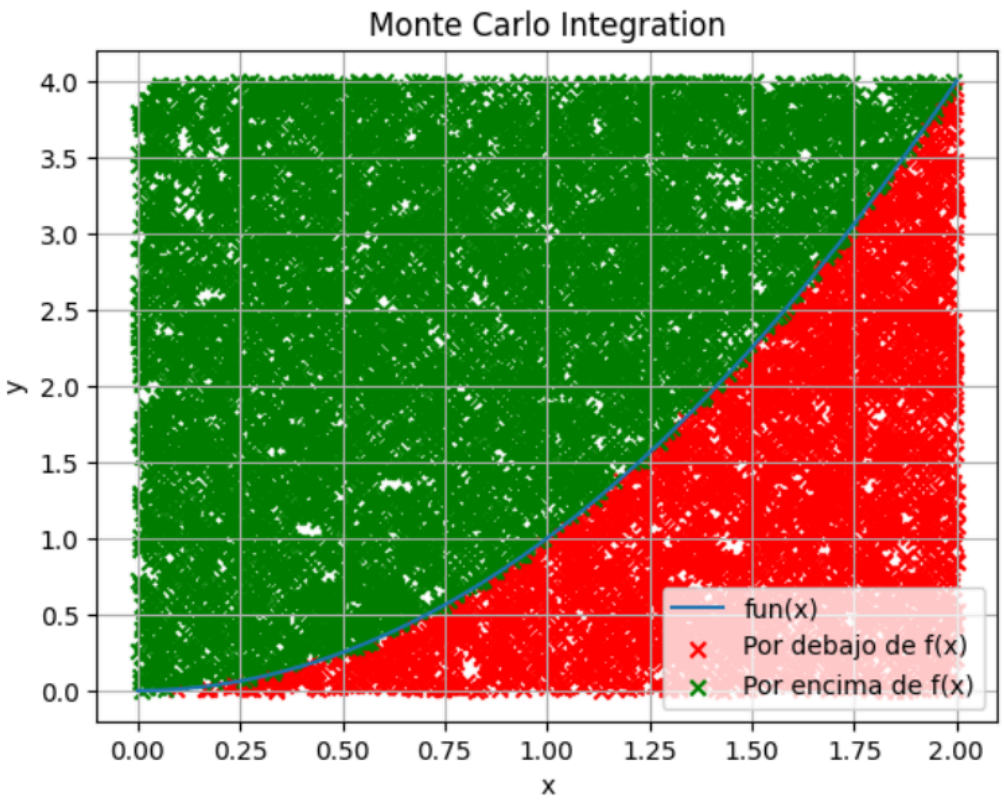
Lo explicado anteriormente, lo he realizado de manera iterativa y de manera vectorial usando la librería numpy. En las gráficas inferiores se puede observar los puntos generados, en rojo los que están por debajo de la función y en verde el resto.

La integral generada por el método iterativo y el método vectorizado es diferente porque son funciones separadas que calculan puntos aleatorios diferentes, aunque el código realiza las mismas operaciones.

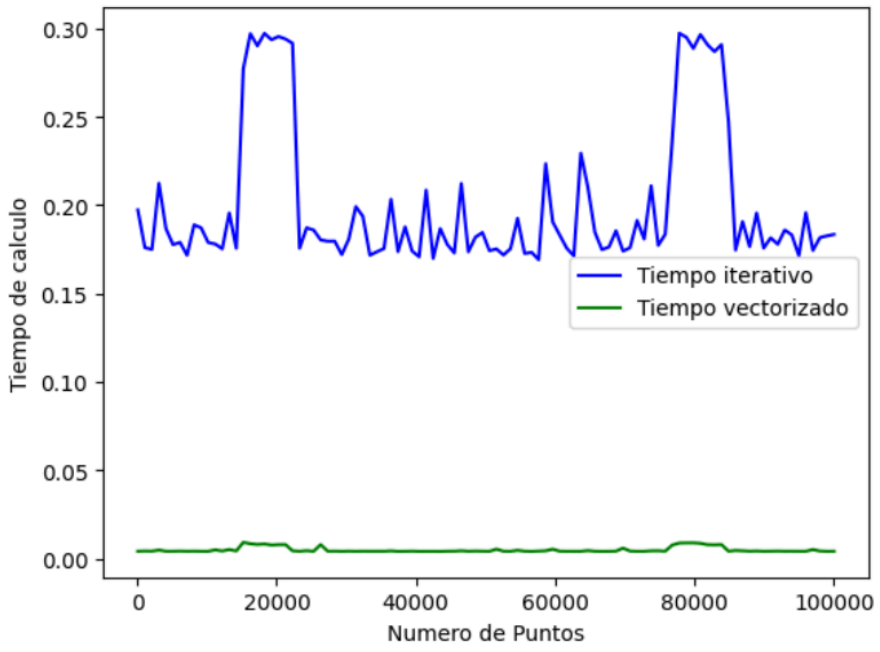
Utilizo también la función de scipy.integrate para asegurar que las soluciones son suficientemente buenas.



```
Integral calculada usando MonteCarlo vectorizado:  2.6784
```



La gran diferencia entre las funciones iterativa y vectorial es el tiempo de ejecución. Como se puede observar en la imagen inferior, el tiempo de cálculo de la función vectorizada es mucho menor.



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

def integra_mc_vec(fun, a, b, num_puntos=10000):

    ##Definimos el espacio
    puntosX = np.linspace(a, b, num_puntos)
    puntosY = np.array([fun(n) for n in puntosX])
    max_y = np.max(puntosY)
    min_y = np.min(puntosY)

    ##Generamos puntos aleatorios dentro del espacio
    x = np.random.uniform(a, b, num_puntos)
    y = np.random.uniform(min_y, max_y, num_puntos)

    ##Numero de puntos que estan debajo de fun(x)
    NDebajo = sum(fun(x) > y)

    return float(NDebajo / num_puntos) * (b - a) * max_y, puntosX, puntosY, x,
y


def integra_mc_iter(fun, a, b, num_puntos=10000):

    ##Definimos el espacio
    puntosX = np.linspace(a, b, num_puntos)
    puntosY = np.array([fun(n) for n in puntosX])
    max_y = 0
    min_y = 100
    for punto in puntosY:
        if punto > max_y:
            max_y = punto
        elif punto < min_y:
            min_y = punto

    ##Generamos puntos aleatorios dentro del espacio
    x = np.array([])
    y = np.array([])
```

```
    for _ in range(num_puntos):
        x = np.append(x, np.random.uniform(a, b))
        y = np.append(y, np.random.uniform(min_y, max_y))

    ##Contamos el numero de puntos que estan debajo de f(x)
    NDebajo = 0
    for i in range(len(x)):
        if fun(x[i]) > y[i]:
            NDebajo += 1

    return float(NDebajo / num_puntos) * (b - a) * max_y, puntosX, puntosY, x,
y

def cuadrado(x):
    return x * x

##Temporizador
def timeStart(func, a, b):
    tic = time.process_time()
    a = func(cuadrado, a, b, 10000)
    toc = time.process_time()
    return toc - tic

##Funcion para dibujar resultados de monteCarlo
def plot_results(puntosX, puntosY, x, y):
    plt.plot(puntosX, puntosY, label='fun(x)')
    plt.scatter(x[cuadrado(x) > y], y[cuadrado(x) > y], marker ="x",
color='red', label='Por debajo de f(x)')
    plt.scatter(x[cuadrado(x) <= y], y[cuadrado(x) <= y], marker ="x",
color='green', label='Por encima de f(x)')
    plt.title('Monte Carlo Integration')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.grid(True)
    plt.show()

print("Introduce el valor de a del intervalo [a, b]")
a = int(input())
print("Introduce el valor de b")
b = int(input())
```

```
print('Integral calculada de Python scipy: ', scipy.integrate.quad(cuadrado,
a, b))
resultado_iter, puntosX_iter, puntosY_iter, x_iter, y_iter =
integra_mc_iter(cuadrado, a, b, 10000)
print('Integral calculada usando MonteCarlo iterativo: ', resultado_iter)
plot_results(puntosX_iter, puntosY_iter, x_iter, y_iter)
resultado_vec, puntosX_vec, puntosY_vec, x_vec, y_vec =
integra_mc_vec(cuadrado, a, b, 10000)
print('Integral calculada usando MonteCarlo vectorizado: ', resultado_vec)
plot_results(puntosX_vec, puntosY_vec, x_vec, y_vec)


##Dibujamos la grafica de diferencia de tiempos entre monteCarlo Vectorial e
iterativo
numpunts = np.linspace(100, 100000, 100)
time_iter = []
time_vec = []
for _ in range(len(numpunts)):
    time_iter.append(timeStart(integra_mc_iter, a, b))
    time_vec.append(timeStart(integra_mc_vec, a, b))
plt.plot(numpunts, time_iter, '-', label="Tiempo iterativo", color="blue")
plt.plot(numpunts, time_vec, '-', label="Tiempo vectorizado", color="green")
plt.xlabel("Numero de Puntos")
plt.ylabel("Tiempo de Calculo")
plt.legend()
```