Práctica 0. Memoria Técnica

Código

import numpy as np

import matplotlib.pyplot as plt

import scipy.integrate as intpy

import math

import time

def funcion\_integral(n\_debajo, n\_total, a, b, M):

    return (n\_debajo / n\_total) \* (b - a) \* M

def iterativo(fun, x, y):

    tic = time.process\_time()

    n\_debajo = 0

    for i,j in zip(x, y):

        if j < fun(i):

            n\_debajo += 1

    toc = time.process\_time()

    return 1000 \* (toc - tic)

def operaciones(fun, x, y):

    tic = time.process\_time()

    num\_debajo(fun, x, y)

    toc = time.process\_time()

    return 1000 \* (toc - tic)

def num\_debajo(fun, x, y):

    a = np.array(y < fun(x))

    return float(np.sum(a))

def dibuja\_flechas(axs, time\_it, time\_fast):

    max\_time\_it = np.amax(time\_it)

    result\_it = np.where(time\_it == max\_time\_it)

    axs[1].annotate(max\_time\_it, xy=(result\_it[0][0], max\_time\_it),  xycoords='data',

            xytext=(0.5, 0.6), textcoords='axes fraction',

            arrowprops=dict(facecolor='black', shrink=0.005), fontsize=12,

            horizontalalignment='right', verticalalignment='top',

            )

    max\_time\_fast = np.amax(time\_fast)

    result\_fast = np.where(time\_fast == max\_time\_fast)

    axs[1].annotate(max\_time\_fast, xy=(result\_fast[0][0], max\_time\_fast),  xycoords='data',

        xytext=(0.5, 0.4), textcoords='axes fraction',

        arrowprops=dict(facecolor='black', shrink=0.005), fontsize=12,

        horizontalalignment='right', verticalalignment='top'

        )

def axis\_lim(axs, a, b, num\_puntos, time\_it):

    plt.xlabel('x')

    plt.ylabel('y')

    axs[0].set\_xlim([a - 0.2, b + 0.2])

    axs[0].set\_ylim([-0.05, 1.05])

    axs[1].set\_xlim([a - 0.2, num\_puntos + 0.2])

    axs[1].set\_ylim([-0.05, np.amax(time\_it) + 0.5])

def coloca\_leyenda(axs, X\_tiempo, time\_it, time\_fast):

    axs[1].plot(X\_tiempo, time\_it, color="red", linewidth=2.5, linestyle="-", label="bucle")

    axs[1].plot(X\_tiempo, time\_fast, color="green", linewidth=2.5, linestyle="-", label="operaciones")

    plt.legend(loc='upper right')

def integra\_mc(fun, a, b, num\_puntos=10000):

    X = np.linspace(a, b, 256, endpoint=True)

    X\_tiempo = np.linspace(0, num\_puntos, num\_puntos, endpoint=True)

    S = fun(X)

    fig, axs = plt.subplots(1, 2, figsize=(12, 5))

    axs[0].plot(X, S)

    M = np.amax(S) # Calculamos el punto más alto de la curva

    time\_it = []

    time\_fast = []

    for size in X\_tiempo:

        x = np.random.uniform(a, b, num\_puntos)

        y = np.random.uniform(a, M, num\_puntos)

        axs[0].scatter(x, y, s=50, c='red', marker="x", linewidth=1.0)

        n\_debajo = num\_debajo(fun, x, y)

        time\_it += [iterativo(fun, x, y)]

        time\_fast += [operaciones(fun, x, y)]

    # Determina la organización de los ejes X e Y en ambas gráficas

    axis\_lim(axs, a, b, num\_puntos, time\_it)

    # Coloca la leyenda de la gráfica del tiempo

    coloca\_leyenda(axs, X\_tiempo, time\_it, time\_fast)

    # Dibuja las flechas que señalan a los puntos más altos de la gráfica del tiempo

    dibuja\_flechas(axs, time\_it, time\_fast)

    # Solución final obtenida con Monte Carlo

    sol = funcion\_integral(n\_debajo, num\_puntos, a, b, 1)

    str\_sol = ("MONTE CARLO --> " + str(sol))

    plt.gcf().text(0.55, 0.95, str\_sol, fontsize=12)

    # Solución final obtenida con la función de integración de scipy

    sol\_buena = intpy.quad(np.sin, a=a, b=b)

    str\_sol\_buena = ("INTEGRAL -->" + str(sol\_buena))

    plt.gcf().text(0.55, 0.9, str\_sol\_buena, fontsize=12)

    plt.show()

    plt.savefig('practica\_0.png')

integra\_mc(np.sin, 0, np.pi, 10000)

# Desarrollo y Resultados

Para poder completar la práctica se ha usado como ejemplos el ejercicio de *Vectorización* proporcionado y ejemplos de la documentación de [Matplotlib](https://scipy-lectures.org/intro/matplotlib/).

Usaremos la Ilustración 1. para explicar el comportamiento de la práctica realizada.

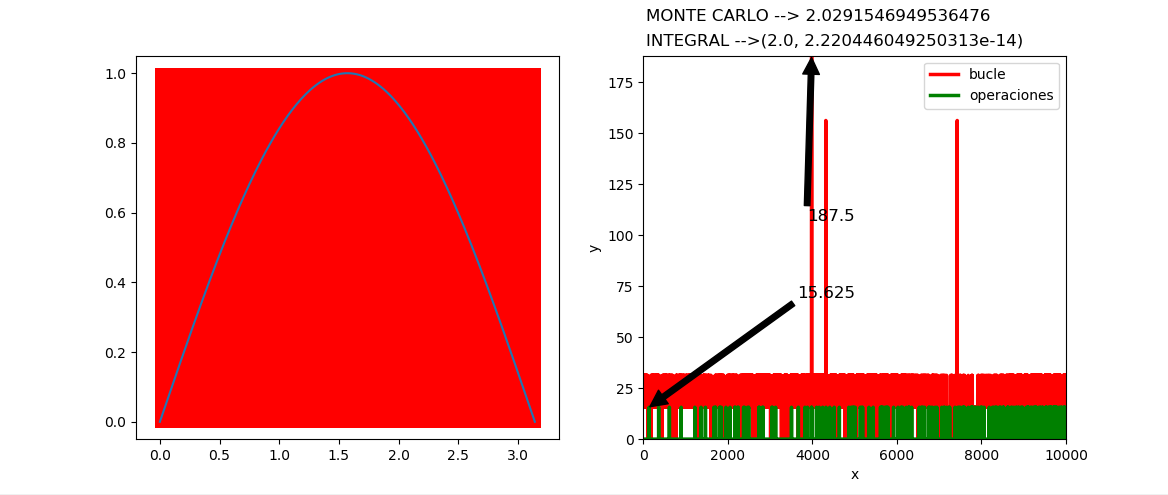


Ilustración 1

Como se puede observar tenemos dos gráficas: la gráfica de la izquierda (A) es en la que hemos creado la función y colocado las “x” de forma aleatoria, y la gráfica de la derecha (B) representa el tiempo que tarda el algoritmo en realizar *num\_puntos* iteraciones (rojo) y el tiempo usando operaciones de vectores (verde).

En la gráfica A se ha usado la función del seno representada de color azul en un intervalo en el eje X de [0, π]. El fondo es rojo debido a la colocación de forma aleatoria en un bucle que va de 1 a 10000, por lo que genera miles de marcas, y por ello se ve de esa forma.

En la gráfica B se pueden observar las dos funciones descritas anteriormente, en la que usando iteraciones se llega a alcanzar un tiempo máximo, este puede variar, de 187.5 milisegundos, a diferencia de el tiempo usando operaciones, el cual no ha variado en ninguna de los tests realizados, de 15.625 milisegundos. La diferencia entre estos dos resultados está en el cálculo del número de marcas que se encuentran por debajo de la función seno (gráfica A).

|  |  |
| --- | --- |
| ITERATIVO | OPERACIONES |
| def iterativo(fun, x, y):      tic = time.process\_time()      n\_debajo = 0      for i,j in zip(x, y):          if j < fun(i):              n\_debajo += 1      toc = time.process\_time()      return 1000 \* (toc - tic) | def operaciones(fun, x, y):      tic = time.process\_time()      num\_debajo(fun, x, y)      toc = time.process\_time()      return 1000 \* (toc - tic)  def num\_debajo(fun, x, y):      a = np.array(y < fun(x))      return float(np.sum(a)) |

Las soluciones a la integral se encuentran arriba de la gráfica B, donde se puede ver el resultado usando *Integración por Monte Carlo* y usando la función *scipy.integrate.quad* de Python. Se puede observar que el cálculo es correcto.

En resumen, se ha comprobado que cualquier función que se pueda realizar con Python o Numpy llevará un tiempo mucho menor que realizando iteraciones para calcular algún resultado.

Alejandro Villar Rubio

Aprendizaje Automático y Minería de Datos, Desarrollo de Videojuegos UCM