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),  xyco
ords='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, linest
yle="-", 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.

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

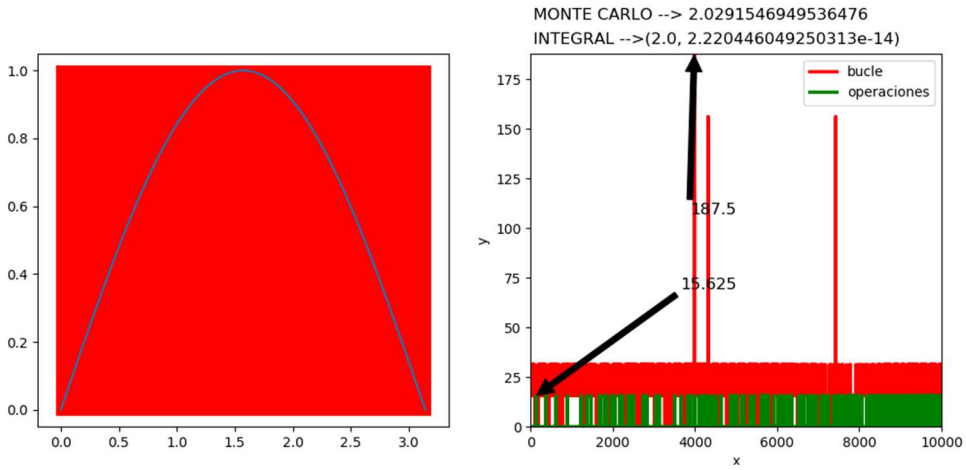

Ilustración 1

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

```
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)
```

OPERACIONES

```
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.