Assignment 6

Lodha Soumya Sachin

EE23B140

1 Objective

To optimize a python code for the trapezoidal rule integration method using cython and compare its performance with equivalent numpy function

2 Cython implementation

```
%%cython --annotate

import cython

# @cython.cdivision(True)

@cython.boundscheck(False)
@cython.wraparound(False)
cdef cy_trapz(f, float a, float b, int n):
    cdef float length, area
    cdef int i

    length = (b - a) / n
    area = 0.0

    for i in range(n):
        area += ((f(a + i * length) + f(a + (i + 1) * length)) * length) / 2.0

    return area
```

By using 'cdef' in place of 'def', Cython treats Python objects as C variables.

C variables are faster because they avoid Python's dynamic type checking, and Cython can generate more optimized machine code for arithmetic and loop operations.

3 Accuracy Testing

```
print(py_trapz(f2,0,np.pi,100000 ))
print(cy_trapz(f2,0,np.pi,100000 ))

x = np.linspace(0,np.pi,100000)
y = np.sin(x)
print(np.trapz(y, x))
```

```
1.9999999998355185
1.9999889135360718
1.9999999998355031
```

```
print(py_trapz(f4,1,2,100000 ))
print(cy_trapz(f4,1,2,100000 ))

x = np.linspace(1,2,100000)
y = f4(x)
print(np.trapz(y, x))
```

```
0.6931471805661945
0.6931451559066772
0.6931471805661955
```

As observed the values are very very similar with the python and numpy trapz function varying only in last 3 decimal places, cython value is also pretty accurate.

4 Performance Testing

This is the performance for squaring x -

```
41.2 ms ± 389 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
25.6 ms ± 440 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
314 µs ± 12.2 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

The order is python, cython and then np.trapz().
Cython does considerably compile in a lesser time but numpy optimizes the function by almost 100 times.
Similar pattern is observed for the rest three functions (please check notebook file for that).

5 Some Remarks

In the python and cython function, most of the time goes in calling the function for every trapezium area calculation twice.
In one single loop we have to call the function twice.

```
for i in range(n):
    area+= ((f(a+i*length)+f(a+(i+1)*length))*length)/2
return area
```

Whereas for the np.trapz() we are just putting list of values of x and y as arguments so we do not need to actually call the function multiple times which optimizes the time by a huge factor.