Assignment 6 Report

EE23B039 A. Lekhasree

Approach

Finding Area Under a Curve Using Trapezoids

To find the area under a curve for a function $f(x)$ over an interval $[a, b]$, this method divides the interval into n parts(considered 1000 parts in all the test cases unless specified) and sums the areas of the trapezoids formed using the following formula:

$$\text{Area of trapezium} = \frac{1}{2} \times (\text{sum of parallel sides}) \times (\text{distance between them}) \tag{1}$$

Python Implementation

The same procedure mentioned above is followed in Python.

Cython Implementation

First, Cython is loaded, and the function is statically typed as in C. A wrapper function, `cy_trapz_test()`, is created to implement this. By statically typing the function in Cython, we can take advantage of C-like performance optimizations. The wrapper function ensures that the correct data types are passed from Python to Cython, helping to prevent type-related errors.

Numpy Implementation

Numpy's built-in function is used directly for this implementation.

Test functions:

Table 1: Integration Results Comparison

Function	Method	Result	Time (s)	Error
$f(x) = x^2$	Python	0.33333349999999984	0.000390	1.6666666652342954e-07
	Cython	0.33333349999999995	0.000124	1.6666666663445184e-07
	NumPy	0.3333335	0.000254	1.66666666689963e-07
$f(x) = \sin(x)$	Python	1.999998355065662	0.001727	0.9999983550656619
	Cython	1.9999983550656624	0.000696	0.9999983550656624
	NumPy	1.9999983550656628	0.001002	0.9999983550656628
$f(x) = e^x$	Python	1.718281971649196	0.001722	1.431901508475164e-07
	Cython	1.7182819716491962	0.000721	1.43190151069561e-07
	NumPy	1.718281971649195	0.000137	1.4319014995933799e-07
$f(x) = \frac{1}{x}$	Python	0.6931472430599359	0.000304	6.249999062735156e-08
	Cython	0.6931472430599374	0.000089	6.24999920706415e-08
	NumPy	0.6931472430599375	0.000130	6.24999921816638e-08

Inferences:

$f(x) = x^2$:

Cython implementation offers the best performance, reducing the computation time to 0.000124 seconds while maintaining a similar accuracy level to Python and NumPy. All implementations show negligible differences in error.

$f(x) = \sin(x)$**:**

Cython again is the fastest with a significant reduction in time to 0.000696 seconds while achieving high accuracy, closely matching the results of the other two methods. Python remains the slowest, with the highest execution time.

$f(x) = e^x$**:**

NumPy implementation provides the fastest execution at 0.000137 seconds, with results closely aligning with those from Python and Cython.

$f(x) = \frac{1}{x}$**:**

Cython delivers the best performance for this function, with the lowest execution time of 0.000089 0.000089 seconds and results closely matching Python and NumPy

Performance Test: Integration of $f(x) = x^2$ from 0 to 10

This performance test involves integrating $f(x) = x^2$ from 0 to 10 using 10 million trapezoids. Below is a comparison of the execution times of three different implementations: Python, Cython, and NumPy.

Table 2: Comparison of Integration Methods

Method	**Result**	**Time (s)**	**Error**
Python	333.3333333333152	3.700722	$1.1102230246251565 \times 10^{-15}$
Cython	333.33333333334724	1.128303	$1.3877787807814457 \times 10^{-15}$
NumPy	333.3333333333349	0.141199	$6.661338147750939 \times 10^{-16}$

The results indicate the following performance improvements:

- The Cython implementation is approximately 3.28 times faster than the Python implementation.
- The NumPy implementation outperforms Cython by approximately 8.00 times, showcasing its superior efficiency for numerical tasks.

Performance improvements achieved by cython function

It runs faster than plain python functions and has the same accuracy as that of python and other implementations of the function.

Comparision with numpy implementation

For f(x) = x^2, sin(x) and 1/x cython ran faster than numpy, but the times kept varying for different executions. For e^x numpy was faster than the other two. From the last part of the question, where 10 million trapezoids were taken, it is seen that as the number of trapezoids increases, the time taken by cython and python increases very much but numpy doesn't vary that much showing that it is the fastest for integration as ideally in integration we consider many small sized trapezoids and this number might shoot up the time taken by python and cython.

Challenges encountered during the Cython implementation and optimization process

Data types while static typing should be taken care.

Some Python modules (like those using dynamic types) are difficult to integrate directly into Cython code, especially if they rely on Python's dynamic typing.

Cython introduces an additional compilation step, which can significantly increase build time compared to regular Python scripts.