Trapezoidal Rule Integration: Analysis Using Python, Cython, and NumPy

Siva Sundar, EE23B151

June 4, 2025

1 Objective

This project focuses on optimizing python code using cython and also comparing the performance of user-defined python and corresponding cython functions, keeping numpy as reference. Accuracy of function's output to reference output is also checked.

2 Methodology

The approach for this assignment involves the following tasks:

1. **Define function which implements trapezoidal rule:** In python, we define py_trapz(f,a,b,n) and in cython, we define cy_trapz(f,a,b,n), where f is the function for which we need to find area in interval [a,b] by splitting them into 'n' trapezoids.
2. **Numpy function trapz() is used to find reference values:** We will have four test cases, and we use numpy.trapz() as the gold standard for finding accuracy for the python and cython functions.
3. **Test cases**: we are asked to find the area of some functions between a particular interval. Table 1 shows the test cases as well as their analytical values:

Test No.	Function	Expected Result
1	$y = x^2$, $x \in [0, 1]$	$\frac{1}{3}$
2	$y = \sin(x)$, $x \in [0, \pi]$	2
3	$y = e^x$, $x \in [0, 1]$	$e - 1 \approx 1.7183$
4	$y = \frac{1}{x}$, $x \in [1, 2]$	$\ln(2) \approx 0.6931$

Table 1: Test cases along with expected results for each function

3 Optimizing the cython function

We can optimize cy_trapz() using specific decorators as well as redefining variables with C-like datatypes. The following optimization steps are used:

- We use '@cython.cdivision(True)', which enables the function to do divisions like a C program, which has better performance than python division.
- We use '@cython.returns(float)' to fix the return value as a float which can reduce overhead due to dynamic typing.
- Data-types of variables are fixed with appropriate types, in function definition as well as local variables used inside the function.

- As this function takes in another function which is used in a loop, for better performance, we pass cython functions. (Wrapped with the same decorators as cy_trapz)
- Using math library from standard C libraries for performance.

```
# Optimized Cython Function
@cython.cdivision(True)
@cython.returns(float)
def cy_trapz(f, float a, float b, int n):
    cdef int i
    cdef float h = (b - a) / n
    cdef float sum = 0.0

    # Initialize sum using the trapezoidal rule
    sum = 0.5 * h * (f(a) + f(b))

    # Loop through the intervals
    for i in range(1, n):
        sum += h * f(a + i * h)
    return sum
```

Figure 1: Yellow lines hint at Python interaction.

to see **jupyter notebook** and the **keyboard layout files**. Instructions on how to run it is provided in the notebook.

4 Results

Results of **Task 4** and **Task 5** are shown in the **jupyter notebook**

5 Observations

- The absolute error is more for cython implementation while less in NumPy and python codes (which are almost equal). This could be the fact that python uses 64-bit float data-type while in cython, I used 32-bit float data-type.
- Python took double the time as compared to Cython and 20 times the time as compared to NumPy.

6 Conclusion

I conclude the following from my observations:

$$Error_{Python} = Error_{NumPy} = \frac{1}{100} \times Error_{Cython}$$

$$Time_{Python} \approx 2 \times Time_{Cython} \approx 20 \times Time_{NumPy}$$

7 References

[1] Cython docs: Importing C libraries in Cython

[2] Cython docs: cython.returns() decorator

[3] Cannot find vcvarsall bat when running a Python script

[4] Microsoft C++ Build Tools