EE2703 : Applied Programming Lab
Assignment 6 - Trapezoidal Rule Integration: Python, Cython and NumPy implementation

Abhinav I S
EE23B002

---

1 Implementation details

1.1 Python Function

1. The step size is defined as $(b-a)/n$ where n is the number of trapeziums
2. Then, we iterate over the number of trapeziums, calculate the area under each trapezium, and sum them to find the total integral.
3. `py_trapz` calls the function to be integrated n+1 times. We introduce the `prev` to store the value of the function at the previous point.
4. Area of each trapezium is $(prev + f(end\_val)) * step\_size/2$, prev is functional value at previous point, $f(end\_val)$ is functional value at current point and $step\_size$ is the width of the trapezium. We do not multiply the width of each trapezium inside the loop, instead, we do so just before returning from the function, to reduce the number of multiplications (increases performance)

1.2 Cython Function

We give type declarations for all the variables used; after this, the implementation is identical to the Python function.

2 Performance and Accuracy Comparisons

- The performance(in terms of time), and accuracy calculations for each of the cases given in the problem statement have been done in the jupyter notebook.
- The cython function usually gives an ~half-time improvement, and numpy is always much better.
- The performance test conducted by integrating x^2 from 0 to 10 using 10,000,000 trapeziums, resulted in a runtime of 1.69s for the python implementation, 565 ms for the cython implementation and 107 ms for the numpy implementation.

- Cython took 33% of the time taken by the python function, and the numpy implementation took 6%
- The accuracy for all the implementations is more or less the same, and this is expected since we are using the same algorithm for all the calculations.

3 Comparison of Cython implementation with the numpy implementation

- The main python interaction in the cython code is when we call the python function for finding the points for the trapezium.
- The numpy implementation takes in arrays, instead of the function, which is better since it does not need to call the python function for each trapezium.
- As a result, the cython function performs worse than the numpy implementation, because of the lack of the overhead in calling the python function many times.
- As an experiment, I tried performing the same analysis, but instead of having a python function return x^2 each time, I wrote a c function using `cython` and it performs much better than the numpy implementation.

4 Challenges faced during Cython Implementation

- Initially, the `cy_trapz` and `py_trapz` functions used to call the function f, 2 times per loop, this was significantly slower because calling the python function more than necessary has performance overheads. So, this was avoided by calling the function once per loop and storing the previous value in a local variable.
- The main problem in optimizing is, to reduce the number of function calls to python function, the way this can be done is to write a c function instead of a python function to get the points.