Assignment 6 - Speeding Up With Cython

Nishanth Senthil Kumar (EE23B049)

October 2024

1 How to run code

- I have submitted a juypter notebook, open it and run the cells sequentially, one by one.
- Ensure the cells are run sequentially, and ensure the function definition cells are run before running the other cells. In case of some error or unexpected behavior, start running the cells from the beginning again.
- I am using timeit to time the function, which is present as a magic function in juypter server, please ensure that it is available.
- I print the value of the integral, followed by the time it takes to run, please wait for a couple of seconds to see the output.
- Please read the instructions at the top of the notebook before running it.

2 Introduction

- I have implemented a Python, Cython, Numpy as per the problem statement, and also a more optimized implementation of the same function in Cython, which gives much better speed ups.

3 Implementation

- The Python version is a simple implementation of the trapezium method, I define the function to be integrated, and call it in py_trapz, the function that implements the trapezium rule.
- The Numpy implementation is using Numpy.trapz(), which requires the data points as Numpy arrays. I have a "generate_data" function that generates the numpy arrays, given a function, and I pass these arrays to numpy.trapz().
- I have made 2 Cython implementations, one as per the problem statement, where the function that to be integrated is an argument to the trapezium rule function, and one where the Cython function finds the integral of a particular function. The first implementation is very slow, as I will explain later with data, and the second one is much faster, beating Numpy.trapz() on average.
- The sine and exponential function have been implemented using math library, and timinig the function is done using the magic function "timeit".
- Apart from the performance test, I have set the step size to 10^4 for all other cases.

4 Comparison of accuracy

- The accuracy of all the 4 method to the known analytical solutions are very good, for the Python and the Cython implementations it depends on the number of steps. For large number of steps, the accuracy is very high (upto almost 7th decimal point).
- This is because all implement the same method and as the step size goes to 0, the value of the summation becomes an integral.

5 Speed comparison

- These times are taken on the Nitin sir's juypter sever.

Table 1: Comparison of Integration Methods for Different Integrals

Integral	**Python**	**Numpy**	**Cython**	**Optimized Cython**
$\int_0^1 x^2\,dx$ (steps: 10^4)	312 μs	27.3 μs	2.67 ms	14.5 μs
$\int_0^\pi \sin(x)\,dx$ (steps: 10^4)	33 ms	27.5 μs	2.63 ms	241 μs
$\int_0^1 e^x\,dx$ (steps: 10^4)	30.2 ms	27.6 μs	2.51 ms	175 μs
$\int_1^2 \frac{1}{x}\,dx$ (steps: 10^4)	28.2 ms	27.2 μs	2.31 ms	19.2 μs
$\int_0^{10} x^2\,dx$ (steps: 10^7)	2.98 s	84.3 ms	2.77 s	14.4 ms

6 Inferences

- We can see that the optimized Cython is the fastest when the function is not implemented using math library, and Numpy.trapz() is the fastest when math library is involved, due to vectorization of operations while using Numpy.
- The normal Cython function performs poorly, being barely faster than normal Python implementation, and sometimes even slower. This is because, for a Cython function to be called using the python interpreter, it has to be declared as a python function("def" is used), and the function whose integral is to be calculated is already a python function. So, effectively, a python function is called inside another python function, and the overheads for this are a lot, so it barely performs better than normal python. The slightly better performance is due to static typing.
- The optimized Cython function is fastest, as it is specialized to perform the integral of a particular function, so I am not calling the function that is to be integrated in the function arguments. As a result, there are no overheads of calling a function multiple times, and this speeds up operations by a lot.
- The comparison between the optimized Cython and Numpy.trapz() is a fair one, as in Numpy.trapz(), we anyways precompute the data points for a particular function beforehand, and then pass it to Numpy.trapz(), so effectively, this too is only for a specific function.
- From a practical perspective, it makes much more sense to implement the optimized Cython function like mine, instead of having a "user-friendly" function where the function itself is an argument. A scientist would much rather prefer a function that performs one task very efficiently and quick, rather than one that is more generalized, but slow.
- Modifying the optimized Cython code is just altering 2 lines of code, along with a function definition (template of which is given in my code) for the new function that is to be integrated.
- I have given a template(of optimised Cython) in the comments on how to declare a different function, it can be used to construct the same for different integrals.

7 Conclusions

- Numpy is fast due to vectorization offered by Numpy arrays, and it works better for functions defined using math library.
- Cython used responsibly can make code very efficient, but if not used properly, the benefits are minimal, and in some cases, negative.