EE2703 Assignment 6: Speeding up with cython

Kaushik G. Iyer (EE23B135)

October 27, 2024

Abstract

In this assignment we were asked to implement the trapz method to perform definite integration on a function using python, cython and numpy. Below contains the analysis of these implementations :)

1 Implementation Details

Mathematically the trapz function is given by:

$$\int_a^b f(x)\,dx = \frac{h}{2} * [f(a) + 2 * \{\sum_{n=1}^{n-1} f(a + ih)\} + f(b)]$$

Where $h = \frac{b-a}{n}$

1.1 Python Implementation: `py_trapz`

1. I basically did a direct translation of the above given formula.
2. I wanted to skip a multiplication so I incremented x in the loop body
3. In order to ensure that I only went through n - 1 points between a and b, I used a for in range loop :)

1.2 Cython Implementation 1: `cy_trapz`

1. I basically did a direct translation of the python implementation.
2. I made sure to use as many cdefs as possible in order to use python minimally.
3. NOTE: This implementation is not ideal as it still contains the overhead associated with calling the python function f (This is handled in the next implementation)

1.3 Cython Implementation 2: `cy_trapz_plus`

1. The actual function body is similar to the first cython implementation. Some differences being that it is now a pure c function that **takes another pure c function (f) as an argument**.
2. In order to this I had to provide custom c functions (for the quadratic, sin etc.),
3. In order to call a c function in python, one has to wrap it first and unwrap it when it is required to be used (This is why the trapz logic was moved to a helper function...The cy_trapz_plus function simply calls this after unwrapping the c function)
4. This overall reduces the amount of interaction with python to a minimum, thus producing very fast speeds :)

1.4 Numpy Implementation 1: `np_trapz`

1. I simply call the np.trapz function here :)
2. NOTE: It is important that for n trapeziums you create the linspace with n + 1. This can be verified by just looking at the graphed example presented in the assignment.
3. This implementation takes the most time, this is due to the overhead of using python functions and also the overhead associated with computing and converting the float value to numpy floats **one by one**. (This is why it is not recommended to use numpy with functions that cannot operate on the entire np array)

1.5 Numpy Implementation 2: `np_trapz_plus`

1. The only difference between this and the previous numpy implementation is that it takes in a function that can **operate on whole numpy arrays**.
2. This allows us to use np function calls (written in c and cpp), which are much faster than the corresponding python functions (When operating on numpy arrays).

2 Analysis of implementations

For the following tests I limited the number of trapeziums to 10_000. The limits of each and the type of function to test are as specified in the assignment.

2.1 Accuracy

The accuracy of all the functions are pretty much the same :) Below I have produced a table of the absolute error percentage given by: $\frac{abs(value-true\_value)}{true\_value}$.

Implementation	**Quadratic**	**Sin**	**Exponential**	**Reciprocal**
Cython 2	4.99e-07%	8.22e-07%	8.33e-08%	9.01e-08%
Cython 1	4.99e-07%	8.22e-07%	8.33e-08%	9.01e-08%
Numpy 1	5.00e-07%	8.22e-07%	8.33e-08%	9.01e-08%
Numpy 2	5.00e-07%	8.22e-07%	8.33e-08%	9.01e-08%
Python	4.99e-07%	8.22e-07%	8.33e-08%	9.01e-08%

2.2 Performance

Since the second cython implementation has the least amount of python overhead, we expect it to run the fastest. The second numpy implementation has the second fastest speed (It is slower than cython since it still has some overhead in creating an array etc.). THe python and simple cython implementation have similar times (with cython being slightly faster sometimes) and the basic numpy implementation takes the longest time (Due to the double overhead as discussed above)

Implementation	**Quadratic**	**Sin**	**Exponential**	**Reciprocal**
Cython 1	1.49 ms	3.05 ms	2.49 ms	1.82 ms
Cython 2	38.5 μs	104 μs	86.7 μs	38.8 μs
Numpy 1	2.32 ms	3.23 ms	3.3 7ms	2.49 ms
Numpy 2	97.7 μs	193 μs	177 μs	100 μs
Python	1.5 ms	3.04 ms	2.76 ms	1.8 ms

3 Performance for large n

I timed the different implementation when asked to compute trapz(quadratic, 0, 1, 1_000_000)

- cy_trapz_plus: 37.6 ms ± 2.39 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

- np_trapz_plus: 207 ms ± 7.52 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
- py_trapz: 1.53 s ± 96.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
- cy_trapz: 1.47 s ± 107 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
- np_trapz: 2.5 s ± 146 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

The second cython implementation is approximately 40 times faster than the base python implementation. It is 5 times faster than the second numpy implementation :).

4 Problems faced [and resolved :)]

The main issue was getting the cython trapz function to take in a c function instead of a python function. (This is the main reason why the basic cython implementation doesn't give that much improvement).

To do this I found out that I had to use the PyCapsule object inorder to wrap the underlying c function (to be used by us later)