Cython Implementation

Deepak Charan S ee23b022

October 2024

1 Introduction

I am Deepak Charan S and this is my report for the fifth assignment of EE2703 course (Applied Programming Lab).

2 Approach:

I first implemented the trapezoidal rule of integration in python (dividing curve into multiple trapezoids and summing their area). I used the same function in Cython but optimised it by defining the datatypes of all the variables. I then checked their performance against each testcase provided

3 How To Run:

- 1. Run the import Cython cell and then the %load_ext Cython cell
- 2. Run the import libraries cell and then run the two functions, py_trapz and cy_trapz
- 3. I have kept separate cells for checking the performance of each function with each of the test cases. You can run any of them/ change the number of iterations and check the time taken to run it (If you want to look at the performance of one method solely, comment the other two %timeit lines)
- 4. I have also kept a separate section to test out the accuracy of python and cython implementation (using numpy's value as reference). You can run whichever test case you like and check its accuracy

DISCLAIMER: I was facing a weird issue in Jupyter wherein I would get NameErrors for not defining functions although I did run them. If you are also facing the same issue, please go the function which cython is using; change function intialsiation line from cdef <datatype> <function name> to just def <function name>.

Once this runs, you change it back to the original format and proceed (It somehow works)

```
cdef double cy_bexp(x): #Cython function for getting e^(value)
    return math.e**(x)

def cy_bexp(x): #Cython function for getting e^(value)
    return math.e**(x)
```

4 Findings:

After all the optimization using Cython, I was able to shorten the execution time by more than 60%.

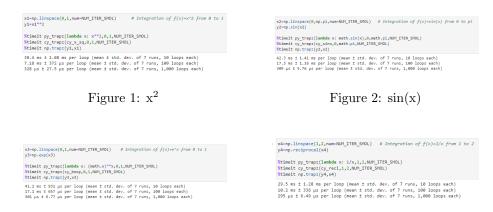


Figure 3: e^x

Figure 4: 1/x

```
x5=np.linspace(0,10,num-NUM_ITER)
y5=x5**2

%timeit py_trapz(lambda x: x**2,0,10,NUM_ITER)
%timeit cy_trapz(cy_x=q,0,10,NUM_ITER)
%timeit np_trapz(y5,x5)

4 s ± 285 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
757 ms ± 62.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
88.4 ms ± 6.84 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Figure 5: x^2 with 10 million iterations

However, numpy.trapz() outshone over both of them by being 10³ times faster (But as number of iterations get larger, Cython and Numpy perform equally well). In terms of accuracy, both python and cython behaved similarly

5 References:

Handout provided in Moodle and info about numpy.trapz()