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 referance). 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%.

```
x1=np.linspace(0,1,num=NUM_ITER_SMOL)      # Integration of f(x)=x^2 from 0 to 1
y1=x1**2

%timeit py_trapz(lambda x: x**2,0,1,NUM_ITER_SMOL)
%timeit cy_trapz(cy_x_sq,0,1,NUM_ITER_SMOL)
%timeit np.trapz(y1,x1)

38.4 ms ± 2.08 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
7.18 ms ± 372 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
328 µs ± 27.5 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Figure 1: x^2

```
x2=np.linspace(0,np.pi,num=NUM_ITER_SMOL)   # Integration of f(x)=sin(x) from 0 to pi
y2=np.sin(x2)

%timeit py_trapz(lambda x: math.sin(x),0,math.pi,NUM_ITER_SMOL)
%timeit cy_trapz(cy_sinx,0,math.pi,NUM_ITER_SMOL)
%timeit np.trapz(y2,x2)

42.3 ms ± 1.41 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
17.3 ms ± 1.16 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
309 µs ± 5.76 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Figure 2: sin(x)

```
x3=np.linspace(0,1,num=NUM_ITER_SMOL)    # Integration of f(x)=e^x from 0 to 1
y3=np.exp(x3)

%timeit py_trapz(lambda x: (math.e)**x,0,1,NUM_ITER_SMOL)
%timeit cy_trapz(cy_bexp,0,1,NUM_ITER_SMOL)
%timeit np.trapz(y3,x3)

41.2 ms ± 931 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
17.1 ms ± 657 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
301 µs ± 6.77 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Figure 3: e^x

```
x4=np.linspace(1,2,num=NUM_ITER_SMOL)   # Integration of f(x)=1/x from 1 to 2
y4=np.reciprocal(x4)

%timeit py_trapz(lambda x: 1/x,1,2,NUM_ITER_SMOL)
%timeit cy_trapz(cy_reci,1,2,NUM_ITER_SMOL)
%timeit np.trapz(y4,x4)

29.5 ms ± 1.28 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
10.2 ms ± 336 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
295 µs ± 6.49 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

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_sq,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 10^3 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()