Assignment-6

Deenabandhan N
EE23B021

Introduction :

- In this experiment , we tried to compare the performances of python , cython and numpy while doing the integration operation using trapezoidal rule.
- The main objective of the experiment was to get familiar with cython implementation and optimising the cython code.
- Let us discuss each methods explicitly in the following sections.

Python implementation :

- I have used be basic lists to define the array of x coordinates and functional values to differentiate it with numpy.
- The computational operation that it takes to perform this integration id=s running a for loop for (n) times.
- I executed this with the timeit command available in the jupyter server for functions and got the following results.
- I have used 1e7 as the number of trapeziums which got divided.

Function	Time taken	Result
$f(x) = x^2$	3.75 s ± 91.2 ms per loop (of 7 runs, 1 loop each)	333.33333333333525
$f(x) = \sin(x)$	4.25 s ± 210 ms per loop (of 7 runs, 1 loop each)	2.000000000000048
$f(x) = e^x$	3.87 s ± 146 ms per loop (of 7 runs, 1 loop each)	1.7182818284590207
$f(x) = \frac{1}{x}$	3.75 s ± 91.2 ms per loop (of 7 runs, 1 loop each)	0.69314718056008765

- I have used math module to define sine and exponentiation functions.

Numpy

- We can infer that numpy has the most optimised version of the code.
- I have used every list as numpy and used pre-defined numpy functions to compute the integration.
- The numpy function to compute the integration based on the trapezoidal method.

Function	Time taken	Result
$f(x) = x^2$	158 ms ± 32.2 ms per loop (of 7 runs, 10 loop each)	333.33333333333525
$f(x) = \sin(x)$	214 ms ± 3.19 ms per loop (of 7 runs, 1 loop each)	1.999999999999982
$f(x) = e^x$	175 ms ± 3.67 ms per loop (of 7 runs, 10 loop each)	1.718281828459046
$f(x) = \frac{1}{x}$	113 ms ± 1.96 ms per loop (of 7 runs, 10 loop each)	0.6931471805599465

Cython

- Cython will convert the python into C code and optimise it.
- I started interpreting the thickness of yellow color from the original code.

```
+03: def f(x):
+04:     return(x*x)
 05:
+06: def py_trapz(f, a, b, n):
+07:     dx=(b-a)/n #defining the infinitesmall length
+08:     int_f=0
+09:     for i in range(int(n)):
+10:         s=(f(a+i*dx)+f(a+(i+1)*dx))
+11:         int_f+=(s*dx)/2
+12:     return(int_f)
```

- From the above picture , I started optimising.
- I optimised by defining the functions in the cpdef definition with its datatype.
- I also optimised by making the for loop into while loop as it reduces the usage of range() function.
- I individually defined the datatype of each variable so that it can interpret on its own.
- I also used the c built functions like sin and exp to run the cython code.

Function	Time taken	Result
$f(x) = x^2$	499 ms ± 4.94 ms per loop (of 7 runs, 1 loop each)	333.3333308085722
$f(x) = \sin(x)$	987 ms ± 12.6 ms per loop (of 7 runs, 1 loop each)	2.000000000000107
$f(x) = e^x$	894 ms ± 5 ms per loop (of 7 runs, 10 loop each)	1.7182818602252408
$f(x) = \frac{1}{x}$	531 ms ± 6.44 ms per loop (of 7 runs, 1 loop each)	0.6931471864029342

Accuracy :

- We can infer that for 1e7 almost all methods perform equally well.
- I have plotted the accuracy of the cython and python method and added it to my notebook.