Assignment 6

Speeding up with Cython

Pure Python Implementation

```
def py_trapz(f, a, b, n):
    dx = (b-a)/(n-1)
    area = 0
    for i in range(n-1):
        x = a + i*dx
        area+=(f(x)+f(x+dx))*dx/2
    return area
```

- This function takes **f** (the function to be integrated), **a**,**b** (the range) and **n** (no. of sample points) and performs trapezoidal integration.

- in each iteration the ith and (i+1)th points are taken and the area of trapezium formed by them is added to the total area

- The function returns the total calculated area

Numpy Implementation

```
def np_trapz(f,a,b,n):
    x = np.linspace(a,b,n)
    y = f(x)
    areanp = np.trapz(y,x)
    return areanp
```

- It creates numpy array(**x**) of sample points using the `linespace` function, and stores the corresponding function values in **y**
- Trapezoidal area is calculated using the builtin in `trapz()` function, which takes **x** and **y** as input
- Returns the total area

Cython Implementation

1. Loading Cython

- `%load_ext Cython` Loads the cython extension to the Jupyter notebook
- `%%cython` cell magic command compiles the python code in the given cell in cython
- `-a` flag gives an annotation of how much of the code translates to **C code**
- `import cython` provides cython decorators for functions
- `cimport cython` allows other cython functionalities like C-level declarations

2. Function definition

```
cdef double cy_trapz(double (*f)(double),double a,double b,int n):
```

- I have defined return value as a C **double** using `cdef`
- The limits are **double** and number of sample points is an **int**
- The input function is a *function pointer* which takes **float** as a parameter and return **float**
- The fixed type definitions makes the function less flexible but faster

3. Defining Variables

- I have used `cdef <Type>` to define each variable including the iterator *i* and area accumulator *area*
- The type of the variable cannot be changed further inside the function

4. Function Decorators

- `@cython.cdivision(True)` : It removes the **Zero-division check** performed by python, making integer divisions faster
- `@cython.boundscheck(False)` : While accessing list elements,Python usually checks for out of bound cases unlike C. This is disabled with this decorator
- `@cython.wraparound(False)` : disables negative array indexing

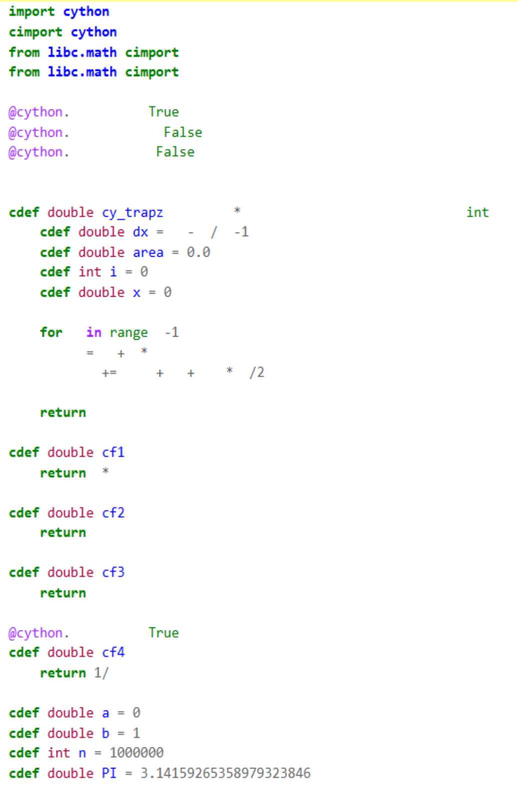
5. Integrand functions

- The integrand functions are also defined in c level for further optimisation.
- Return value and input parameter are defined as double
- For *sin* and *exp* functions I have used the **libc** libary which replicates c functions

6. Evaluation

- The functions are called with c-type variables, `cy_trapz(cf1,a,b,n)`
- I have calculated the execution time using `time` libary in python. I couldn't use %%timeit as it cannot be called within a cython block
- The calculated area and the time taken are printed for each function

- All lines are in the cythonn annotation are white (no yellow lines) indicating maximum optimisation



```
import cython
cimport cython
from libc.math cimport
from libc.math cimport

@cython.          True
@cython.            False
@cython.           False


cdef double cy_trapz         *                              int
    cdef double dx =    -  /  -1
    cdef double area = 0.0
    cdef int i = 0
    cdef double x = 0

    for   in range  -1
          =   +  *
            +=      +   +     *  /2

    return

cdef double cf1
    return  *

cdef double cf2
    return

cdef double cf3
    return

@cython.          True
cdef double cf4
    return 1/

cdef double a = 0
cdef double b = 1
cdef int n = 1000000
cdef double PI = 3.14159265358979323846
```

7. Comparision

Function	Limits	Sample points	Pure Python	Numpy	Cython
x*x	(0,1)	1e6	217.3 ms	16.54 ms	7.49 ms
sin(x)	(0,PI)	1e6	1667.36 ms	23.21 ms	29.19 ms
exp(x)	(0,1)	1e6	1608.81 ms	20.56 ms	16.55 ms
1/x	(0,1)	1e6	237.82 ms	14.82 ms	6.16 ms
x*x	(0,1)	1e7	2335.81 ms	156.46 ms	62.89 ms

- Looking at the latency of pure python, numpy and cython implementations simulateously, it is clear that:
  i. Cython is **much faster** than pure python
  ii. Cython is **considerably faster** than numpy function

- When we compare the output value of the area, all three implementations give the same value upto **11 decimal places**

- Hence Cython optimises the calculation without comprimising on the accuracy

NOTE: The report has extended to three pages due to the attached images and codes, and the formatting.