1 Assignment 5

1.1 Python implementation of the heat equation

In the first part of the assignment, I only used a list ,which is a native datastructure in python and may contain items from different types, and nested loops. The implementation was quite easy. However when I increase the size ($M \times N$) of a rectangle or *t1* it takes more time to be solved. Python does not use the contiguous memory for lists, so accessing to each element in a list become costly for the processor.

Based on the setup from the assignment's description my implementation has been executed in 54 seconds.

1.2 NumPy and C implementations

1.2.1 Numpy

The second implementation was done using the Numpy module. Converting the nested loops from the previous section and using a new syntax can solve a heat equation much faster. However, after I managed to convert the plain solver to the numpy version, the result was simpler than I would have expected.In the Numpy version I only used one loop which runs between [*t0*,*t1*].

```
loopCounter=t0
firstRow=1
firstCol=1
lastRow=n-1
lastCol=m-1
while(loopCounter<=t1):
    u[firstRow:lastRow,firstCol:lastCol]=u[firstRow:lastRow,firstCol:lastCol] + dt \
    * (nu*u[firstRow-1:lastRow-1,firstCol:lastCol] \
    + nu*u[firstRow:lastRow,firstCol-1:lastCol-1] \
    - 4*nu*u[firstRow:lastRow,firstCol:lastCol] \
    + nu*u[firstRow:lastRow,firstCol+1:lastCol+1] \
    + nu*u[firstRow+1:lastRow+1,firstCol:lastCol] \
    + f[firstRow:lastRow,firstCol:lastCol])
    loopCounter+=dt
return u
```

The numpy solver can work out the heat equation in 2.06 seconds. This is because the numpy arrays are saved using contiguous memory allocation.

1.2.2 Instant

In this version I declared a function in pure C language and called that via Instant module. As a result it took only 289 ms to execute the function. This is why I did not to use any python datatypes in my nested loops.

In addition, I declared a *inline_with_numpy* function that takes 3 arguments. Namely, c_code provided with a string variable which contains a whole C code, arrays which is get 3 arrays u,f and args(I have decided to use the 3 arrays, which in third one I can save the *t0*,*t1*,*dt* and *nu*).

```
call_func = inline_with_numpy(c_code, arrays =
                        [['x1', 'y1', 'u'],['x2', 'y2', 'f'],['x3','args']],
                        cache_dir="_cache")
```

The downside is that debugging the code is more difficult. However, it can be more useful when we only want to rewrite small piece of code in C.

1.3 Testing

For the testing part, I have used the py.test to solve the heat equation via the numpy solver. In addition, I have calculated the value of the analytic_u and f in the nested loops structure. The error value for the (100×50) rectangle is less than 0.0012 and when I increase the size the error will decrease accordingly.

1.4 User interface

All functions that I have created for the user interface are included in heat_equation_ui.py. I have defined *timefunc* funcion that is a wrapper for calling a function with the *timeit* module.If we want to measure time for a function with non-primitive arguments, it becomes necessary to call it within a wrapper function.

I also defined two extra functions ,writeOnDisk and loadFromDisk, that are responsible for loading an initial matrix and saving the result on the disk.

I have used the if-else statement in my function for checking a input parameters,optional and non-optional, which caused a little mess in my code.

1.5 Swig and Cyton

1.5.1 Swig

For this part I have decided to use the *SWIG*. To harness the *SWIG* power, first I had to write a interface file that act as a translator between Python and C:

```
/* solver.i */
%module solver
%{
        #define SWIG_FILE_WITH_INIT
        #include "solver.h"
%}
%include "numpy.i"
%init %{
        import_array();
%}
%apply (int DIM1, int DIM2, double *INPLACE_ARRAY2) {(int x1, int y1, double* u)}
```

```
%apply (int DIM1, int DIM2, double *IN_ARRAY2) {(int x2, int y2, double* f)}
%include "solver.h"
```

As you can see I have used *INPLACE_ARRAY2 that is a provided keyword in SWIG which help us to declare a two dimensionals array as an Input and output.For the second array I only used *IN_ARRAY2 which become handy for stating an two dimensional array as input.

The speed for this function was 1.3 s.

1.5.2 Cython

The Cython implementation is similiar to function that I have wrote in python, There are two main differences between the plain python and Cython in my implemenations:

Firsly, I have used the *ndarray[double, ndim=2]* which is used for declaring two dimensional array.

Secondly, by decalaring a variables ,specially loop's counter, with cdef I have gained more speed.

The speed for this function was 851 ms.