Peer-review of assignment 4 for INF3331-henrijn

Torbjørn Vik Lunde, torbjovl, torbjovl@uio.no

Eirik Haatveit, eirish, eirish@uio.no

Christoffer Kolbjørnsen, chriswko, chriswko@uio.no

October 12, 2016

We used Python 3.5.2 on Red Hat Linux 7.2

General feedback

No readme! That makes it difficult to figure out how to run your scripts. We used 30 minutues of reverse engineering to understand which parameters we needed to get a sensible picture.

In general it looks like you are making things difficult for yourself with colour. We recommend using matplotlib for color drawing. It can take an array of ints and lets you with two lines specify a colour scheme.

```
plt.imshow(yourArray, 'inferno')

plt.imsave('filename.png', yourArray, cmap='inferno')
```

In general we would recommend splitting your code more up into different functions.

Assignment 4.1: Python implementation

Doesn't run as an independent script. To make a script that is also a module, run you can use something like this:

```
if __name__ == "__main__":

makeMandelbrotImageTraditional(-1, 1, -1, 1, 0, 0, "foo", "file.png")
```

Assignment 4.2: numpy implementation

As you say, this one is slower.

The point of numpy is to use vector operations. This means that instead of thinking of operations on individual numers you do operations on a bunch of numbers in parallel.

This means that instead of iterating through each element in the (2d) array, you want to add arrays toghether.

As an example, let's say you have one numpy ndarrays of the numbers 1 to 9 called *nums*. To get an array where you double them you would do this:

```
doubleNums = nums * 2
```

This would in one fell swoop double *every* number in nums.

So in your example, you could do this inside a loop to get a very fast calculation of many values at the same time:

```
z *= z # squares all values in z

z += c # adds all values at corresponding positions at c to z
```

Assignment 4.3: Integrated C implementation

The C version is actually not as hard as you might think. You have to type decorate (like you do in Java) and compile (like you do in Java). After that you simply have to write a very simple set of loops. The Cython compiler is smart and if your code is simple enough it can really make it go fast.

For example:

```
cpdef int i

cpdef np.ndarray[int, ndim=1] doubleNums

for i in range(10000):

duobleNums[i] = i * 2
```

This would go really, really, REALLY fast!

Assignment 4.5: User interface

It's ok.

Would be nice if values was optional (which means using named arguments).

This is easy to do with argparse, which would also make other things a lot easier. It should also deal with invalid values more easily.

Assignment 4.6: Packaging and unit tests

You should test the module by importing it and running it, not copy pasting the code. We were able to install and use the module though. Great.

The group could not agree on specifically what this part of the assignment asks so we are unable to decide how correct it is (aside from the above comment).

Assignment 4.7: More color scales + art contest

Different colours work!

Assignment 4.8: Self replication

Works! (aside from 4 extra newlines)