Peer-review of assignment 4 for *INF3331-jostbr*

Sigbjørn Eide, sigbjoei, sigbjoei@uio.no
Nicolai August Hagen, nicolhag, nicolhag@uio.no
Anton Abilov, antonab, antonab@uio.no

October 15, 2016

1 Review

Python version and operating system used:

Python 3.5, OS X.

General feedback

You have thorough pydoc documentation. Since you use triple-quotes for your docstrings, you can easily split them into multiple lines to increase readability. According to the pep8 standard (as well as Atom recommendations), lines should not be longer than 79 characters. Otherwise, you use good and self explanatory variable naming in your programs.

To increase readability in your program, you should avoid making multiple variable definitions in *one line*.

Kudos for showing execution time when calling directly upon mandelbrot_*.py

Assignment 4.1: Python implementation

The code works as expected, and everything is answered. Nice and readable code with good use of docstrings. Sweet as!

Assignment 4.2: numpy implementation

Your implementation works as expected and everything is answered. There's good use of numpy with vectorization and numpy arrays. The numpy implementation is significantly faster than 4.1. Awesome dude!

Assignment 4.3: Integrated C implementation

When you are declaring variables with cdef, you could use it as a function and write several declarations with one cdef: as follows:

```
cdef:
        double delta_x = ...
        double delta y = ...
```

The code works as expected, and the cython implementation is sufficiently faster than 4.2

Assignment 4.4: An alternative integrated C implementation

Some of the lines of code in the c-file is pretty self explanatory, and should not have a comment also.

If possible you should avoid too many nested statements, as this decreases the readability of your code. In this case it could be solved by putting some of the nested statements in a function an calling it from the for loop.

You have very good runtime and once again, the swig implementation is faster - good!

Assignment 4.5: User interface

Good documentation. Some parts of the code are hard to read, for example in the following part:

```
    if (input("\nDo you want to enter custom parameters (if not, default is used) [y/n]? ").rstrip() == "y"):
            for arg in self.args:               # Ask user for value for each parameter
                value_registered = False

                while (not value_registered):   # Keep asking user, per parameter, until valid type entered
                    if (arg in self.args[0:4]): # The parameters of type float
                        try:
```

One reason for making this nesting necessary is that you have to check for many different kinds of cases. For a next time, I think the code complexity could have been avoided by using a command-line parser such as argparse or getopt.

Apart from that, the code is working as expected and your own implementation of argument parsing handles invalid inputs correctly. Well done!

Assignment 4.6: Packaging and unit tests

We think that the packaging of your modules is a bit too complex, and you could have simplified it by having one Mandelbrot package that includes numpy, python, swig and cython modules, instead of nesting them as their own packages.

Your tests are well documented and both pass.

Assignment 4.7: More color scales + art contest

Awesome contest image! Well implemented and documented visualize_mb module that allowed us to experiment with some interesting mandelbrot plots.

Assignment 4.8: Self replication

Creative solution, but your code is actually not self-replicating, since you forgot to add the colon at the end of one of your prints, specifically:

```
"for current_line in list_of_lines[7:]"
```

This also illustrates a problem with your approach being only able to replicate exactly what's in the file, and if you add more code, you would have to add that in the print statements as well. A suggested solution would be the following:

```
import inspect
import sys
# sys modules accesses the current definitions
# inspect inspects "live objects", so we aren't reading the file
print(inspect.getsource(sys.modules[__name__]))
```