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

Reviewer 1, thomagb, thomagb@student.matnat.uio.no
Reviewer 2, aasmunkv, aasmunkv@student.matnat.uio.no
Reviewer 3, haavaoy, haavaoy@student.matnat.uio.no

October 16, 2016

1 Review

System used for review: Mac OSX 10.11.6, Python 3.5.2

General feedback

Generally the implementations work as expected and produce nice images of the mandelbrot set. I have some general comments regarding documentation and code style in the section about assignment 4.1, which mostly apply to all the four implementations.

Assignment 4.1: Python implementation

Is the code working as expected?

The code works as expected and produces a nice image of the mandelbrot set.

Is the code well documented?

You could maybe have documented the code a bit more. Docstrings with a description of all the input arguments and return values are nice, and makes it easier to read the code and understand the functions. I find the Numpy style docstrings to be an easy and clean way to document Python code. There are some good examples here `https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_numpy.html`

Is the code written in Pythonic way?

The structure of the script seems sensible, with good use of functions.

There are some places where the code is not in line with the PEP 8 style guide.

- Spaces around keyword/parameter equal signs (see line 32).
- Missing whitespace after ',' (see line 25).
- Line is longer than 79 characters (see line 25, 29 and 32).

Also I like to add a few empty lines to separate blocks of the code and make the code little more readable.

I get a warning from matplotlib regarding the plotting.

```
UserWarning: The backend (<class 'matplotlib.backends.backend_macosx.RendererMac'>)
does not support interpolation='none'. The image will be interpolated with
'nearest` mode.
  "mode." % renderer.__class__)
```

You should probably remove the `interpolation` argument from `imshow`.

Assignment 4.2: numpy implementation

Is the code working as expected?

The code works, and I get a plot. But as you yourself point out it takes significantly longer to compute than the implementation in 4.1. Also I got two warnings:

```
mandelbrot_2.py:15: RuntimeWarning: overflow encountered in square
  computations = computations**2 + c
mandelbrot_2.py:15: RuntimeWarning: invalid value encountered in square
  computations = computations**2 + c
```

So clearly the implementation is not optimal. The solution does not make use of vectorization, which explains why it is not faster than the pure Python implementation in 4.1.

I'm not entirely sure how your implementation works, but there seem to be a lot of loops, which when using Numpy is a bad thing.

Here is an example of how you could use vectorization in your implementation:

```python
def mandelbrot_calc(stepLimit, startX, endX, startY, endY, resolution):
    Nx = (endX - startX) / resolution
    Ny = (endY - startY) / resolution
    real = np.linspace(startX, endX, Nx)
    imag = np.linspace(startY, endY, Ny)

    x, y = np.meshgrid(real, imag)
    c = x + y*1j

    z = np.zeros((Ny, Nx))
    values = np.zeros((Ny, Nx))

    for i in range(stepLimit):
        index = (np.absolute(z) > 2)
        img[index] = i
        z[index] = 0
        c[index] = 0

        z = z*z + c

    return values
```

Is the code well documented?

I would have liked a few more comments and docstrings for the functions. See my comments for assignment 4.1 for more detail.

Is the code written in Pythonic way?

More or less. I have the same comments as for assignment 4.1.

Assignment 4.3: Integrated C implementation

Is the code working as expected?

Yes, the code works as expected and produces a plot.

Is the code well documented?

I have the same comments as for assignment 4.1.

The Cython code has no comments at all. The code is so simple that it's still easy to read and understand, but a few comments would have been nice.

Is the code written in Pythonic way?

Again the same comments as for assignment 4.1.

Can you find unnecessarily complicated parts of the program?

Like assignment 4.2 you have unnecessary loops in your Python code. Instead of looping over every point in the image in the `mandelbrot_calc` and `compute_mandelbrot` functions, you could do this in the Cython code. This should significantly reduce computation time, and would probably make the code easier to read as well.

I don't understand why you have both a `compute_mandelbrot` and `mandelbrot_calc` function. It seems like the functionality is more or less exactly the same. The `compute_mandelbrot` function I guess is needed for the command line interface or packagaging in assignment 4.5 and 4.6, but then `mandelbrot_calc` seems unnecessary.

There are some errors in `compute_mandelbrot`, but I will comment that in the section on assignment 4.6, since that's where the function is used.

Assignment 4.4: An alternative integrated C implementation

Is the code working as expected?

Yes, the code works as expected and produces a plot.

Is the code well documented?

For the Python code I have the same comments as for assignment 4.1.

The Cython code has no comments at all. The code is so simple that it's still easy to read and understand, but a few comments would have been nice.

Is the code written in Pythonic way?

Again the same comments as for assignment 4.1.

Can you find unnecessarily complicated parts of the program?

As in assginment 4.3 the loops in `mandelbrot_calc` are unnecessary. If you instead did all the loops in C, the computation time should be significantly reduced.

I struggled to figure out how to return an array from C to Python using SWIG. But ended up using Numpy.i, a SWIG interface for Numpy. `https://docs.scipy.org/doc/numpy/reference/swig.interface-file.html`

There was a small error in the `mandelbrot_4_setup.py` file. I had to change line 17 from

```
sources=['mandelbrot_4_swig_wrap.c', 'mandelbrot_4_swig.c'],
```

to

```
sources=['mandelbrot_4_swig.i', 'mandelbrot_4_swig.c'],
```

in order to be able to build the implementation.

Assignment 4.5: User interface

Is the code working as expected?

The code seems to work as expected. I have tried different command line options, everything works fine. As far as I can tell the user interface also handles invalid inputs well.

Is the code well documented?

Again some of the same comments as in assignment 4.1.

Is the code written in Pythonic way?

Mostly I don't have much to comment here. Code is fairly easy to read and understand.

Instead of writing all of the command line interface yourself it might be easier to use a package like argparse. This would probably save some time when writing the code, and makes it easier for others to read and understand. Also makes it easier to handle errors in the user interface.

In the `--help` instructions it would have been nice to know which implementation corresponds to which number.

Assignment 4.6: Packaging and unit tests

Is the code working as expected?

I am not able to install the package correctly. After installing using

```
python3 setup.py install
```

I tried to import the package as `mandelbrot`, but get the error:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python3.5/site-packages/mandelbrot.py", line 11, in <module>
    import mandelbrot_1 as m1
ImportError: No module named 'mandelbrot_1'
```

The problem is that you import `mandelbrot_1`, `mandelbrot_2` and `mandelbrot_4`, but they are not installed as modules.

You could either make sure you install all the four implementations in your package. For instance by moving all the relevant scripts to a subfolder *mandelbrot* and then install using `packages=['mandelbrot']` as an argument to `setup` instead of `py_modules=[name]`.

Or you could split the command line interface part and the module part into two different files, so that you don't import the implementations you don't use in your module when it is installed.

Also you install `mandelbrot_3.py` as a command line script, which seems unnecessary, since I don't think `mandelbrot_3.py` is meant to be called from the command line.

The test to see if the rectangle is entirely outside the mandelbrot set is not correct. I tried to compute the set for the rectangle $x \in [-2, 1]$, $y \in [-1, 1]$, which does cover the mandelbrot set, but the function returns false.

`compute_mandelbrot` from `mandelbrot_3.py` has two small errors which prevent the function from executing successfully:

- `linspace` is used, but is not imported
- in line 24 `arange` is used, when it should be `linspace`

After fixing those, the function seems to works as expected.

Is the code well documented?

See comments for assignment 4.1

I like that `compute_mandelbrot` has a comment wich explains its functionality.

Is the code written in Pythonic way?

See comments for assignment 4.2

Programming part not answered

As far as I can tell there are no unit tests.

Assignment 4.7: More color scales + art contest

The script accepts different color maps. Very nice! I don't have any more to comment here.

Assignment 4.8: Self replication

Not much to comment here. This works exactly as expected. Good work!