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

Jieun Jeong, jieunj, jieunj@ifi.uio.no
John Hald, johnhald, johnhald@ifi.uio.no
Sebastian Benavides, sebasben, sebasben@math.uio.no

October 13, 2017

Review

Python 3 and Ipython were used for the following review.

General feedback

- A very good instruction on how to unpack your own implementation, together with trouble shooting, and how to run the scripts.
- Comments, docstrings and explanation are well provided in general.
- Everything runs well.
- All the tests are executed at one go.
- All in all, good job!

Assignment 4.1

The functions, `test_integral_of_constant_function` and `test_integral_of_linear_function`, do what they are supposed to do. Your code is written in a clean way, as well as the docstrings. You could have left out the parts where you manually defined the F functions by yourself since the expected answers are given as 2 and 1, respectively to each function, in the exercise text.

Assignment 4.2

The function `integrate` works well. The variable names are easy to recognize. Docstrings are missing in `integrator.py`, but there was no problem in understanding your program. You did not need to use the function `float()`, though, because the returned value would be handled to be the instance of float anyways. The program could have handled some invalid inputs, for example, if `b` is larger than `a`, in the following way:

```
    def integrate(f, a, b, N):
         width = abs(b-a)/N   # The step size should be positive
         areal = 0
         for i in range(N):
             areal += f(a + i*width)
         return (areal * width)
```

The test of your code:

```
In [10]: f = lambda x:x          # F = 1/2 * x^2, F(1)-F(0) = 0.5
In [25]: Integrator.integrate(f, 0, 1, 100)
Out[25]: 0.495
In [26]: Integrator.integrate(f, 0, 1, 1000)
Out[26]: 0.4995
In [27]: Integrator.integrate(f, 0, 1, 10000)
Out[27]: 0.4999499999999991
```

The plot should show how the error decreases while N increases. The x-axis should be N, and the y-axis should be the difference between the expected result, $\frac{1}{3}$, and the actual result. Therefore, the plot should be descending.

The pseudo code for plotting:

```python
import numpy as np
import matplotlib.pyplot as plt

N = []   # Your own number of points of N
f = lambda x:x**2
expected = 1/3
error = [abs(expected - integrate(f, 0, 1, n)) for n in N)]
x = np.array(N)
y = np.array(error)

# Design the plot as you want

plt.plot(x,y)
plt.show()
```

Assignment 4.3

As in 4.2, it is well-written and does what it is supposed to do. Numpy is used effectively, and the report proves it (runtime with $N = 100000000$: 33.3 s with pure python implementation, 1.6 s with numpy implementation). The name of the function could have been `numpy_integrate` as in the exercise, so that it is not confusing with the other `integrate` functions. Some docstrings could have been provided.

Assignment 4.4

It is clean and well adopted. The report shows that your numba implementation was successful and efficient. Your comments in `numba_integrator.py` cites the source of your implementation, which is important. Though, you could have written better docstrings based on what you have written so far. Also, the answers to some questions in the exercise are missing in the report.

One minor thing to point is that your comments in `test_integrator.py` states "Printing result to report3.txt" for all different reports under the function `test_integral_task_42_43_44_45`. It was a bit confusing to find where your program tests which integrator at first glance.

Assignment 4.5

The code is well written and easy to understand. It runs and does what it is supposed to do as well. It is observed from the tests and the report that your cython implementation improves the performance of the pure python implementation by far. Some docstrings could have been given.

Assignment 4.6

All the `midpoint_integrate` functions are cleanly written and runs well. However, it is unclear how you found the lowest Ns for integrators. The function `test_integrator_comparison` does show all the tests are successful with the given Ns, but you could have explained how you reached to the conclusion that those numbers were optimal.

As mentioned earlier, you could have not used the function `float()`. Some inconsistency has been observed as I reviewed all files step by step. In some functions, you used the number 2 as integer or float, or passed it to a variable named `tall` to find the midpoints. Also, the step size is called `width` or `h` in different places. They could have had consistent names.

Assignment 4.7

As pointed in general feedback, this part is flawless.

Assignment 4.8

N/A