Peer-review of assignment 4 for 3331-mathiaki

Reviewer 1, hanshsa, hanshsa@student.matnat.uio.no

Reviewer 2, gautesol, gautesol@student.matnat.uio.no

Reviewer 3, viktorro, viktorro@student.matnat.uio.no

Introduction

Goal

The review should provide feedback on the solution to the student. The main goal is to give constructive feedback and advice on how to improve the solution. You, the peer-review team, can decide how you organise the peer-review work between you.

Guidelines

For each (coding) exercise, one should review the following points:

- Is the code as expected? For non-internal functions (in particular for scripts that are run from the command-line), does the program handle invalid inputs sensibly?
- Is the code documented? Are there docstrings and are the useful?
- Is the code written in a Pythonic way? Is the code easy to read? Are the variable/class/function names sensible? Do you find overuse of classes or not sufficient use of functions or classes? Are there parts of the program that are hard to understand?
- Can you find complicated parts of the program? If so, suggest an improved implementation.
- List the programming parts that are not answered.

Use (shortened) code snippets where appropriate to show how to improve the solution.

Points

The review is completed by pushing the review Latex source file (.tex files) and the PDF files to each of the reviewed repositories. The name of the files should be *feedback.tex* and *feedback.pdf* in the students assignemnt4 directory.

You will get up to 10 points for delivering the peer-reviews. Each of you should contribute to the review roughly equivalently - your team will get the same number of points. (In case a team-member does not contribute, please email simon@simula.no.

Review - to be filled out

Python version: 3.6. Operating system: OSX and Windows 10.

General feedback

You test all implementations, and they all pass. Some unnecessary typos, ref. 4.1. Should improve readability somewhat.

You have docstrings for most integration functions, but not all. Most of the functions are already documented in the handout, but we liked that you included docstrings because docstrings can make the most important information more accessible.

Assignment 4.1

You have a typo in your test.

```
from polynomilas import Polynomial as Polynomial
```

But after fixing this, all tests run.

Assignment 4.2

Clean and readable code. Except the error_func function.

Your code:

```
c=[2**(i+1) for i in range(22)]

start = timer()
for i in c:
    val=integrate(f,a,b,i)
    val_list.append(val)
    err_list.append(abs(val-true_val))
```

Our suggestion:

```
N_values = [2**(i+1) for i in range(22)]
# Changed name of c to N_values, to make description more specific
# Added space before and after "="-sign in assignment

start = timer()
# Changed incrementer-name to N and updated integrate function
for N in N_values:
    val = integrate(f,a,b,N)
    val_list.append(val)
    err_list.append(abs(val-true_val))
```

Added spaces according to recommendations from PEP 8:

"Always surround these binary operators with a single space on either side: assignment (`=`), augmented assignment (`+=`, `-=`etc.), comparisons (`==`, `<`, `>`, `!=`, `<>`, `<=`, `>=`, `in`, `not in`, `is`, `is not`), Booleans (`and`, `or`, `not`)."

Assignment 4.3

Interesting numpy-solution, regarding the try/except to handle input functions returning a constant

(by indexing the return value of `f(points)` to check that it's a numpy array). Easy to read integration, and successful vectorization.

Assignment 4.4

Two unused import statements; `autojit` and `plt`. Otherwise, good. Nice that the function accepts a python-function as a parameter f. You have redundant parameters in your wrapper function.

```
def midpoint_numba_integrate(f,a,b,N):
    f=jit(f)
    @jit(nopython=True)
    #a, b and N are redundant
    def wrapper(a, b, N):
        dx=(b-a)/(N)
        retsum=0
        half=dx/2
```

Assignment 4.5

Looks good. You got a respectable speedup.

Assignment 4.6

Easy to see the correlation between your integrate and midpoint integrate functions. Very readable.

Assignment 4.7

You did not make an installable Python package that included the integrate functions and tests. However, the task was extremely ambiguous, with the header "Making a module" (in Python, a module is simply a Python file). We liked your `__call__` function interface.

An improvement to the `__call__` function would be to raise an Error in case of illegal arguments, instead of returning False. This is more common and preferred in Python - see PEP 20: "Errors should never pass silently.".

Pro tip: The type of the empty list (and other lists) is simply `list`. Therefore, you can simplify:

```
if type(f)==type([]):
```

to:

```
if type(f) == list:
```

Assignment 4.8

Participated. Good luck.