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

Reviewer 1, henrikvgrikvg, henrikvg@student.matnat.uio.no
Reviewer 2, eebirkel, eebirkel@ifi.uio.no

October 13, 2017

1 Introduction

1.1 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 organize the peer-review work between you.

1.2 Guidelines

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

- Is the code **working 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 **well documented**? Are there docstrings and are the useful?
- Is the code written in **Pythonic way** [1]? 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 **unnecessarily 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.

1.3 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[2].

2 Review - *to be filled out*

Specify the system (Python version, operating system, ...) that was used for the review.
Python 3.5.2 via cmd on Windows 10.

General feedback

Use this section to give general feedback about the solution such as advice for improved programming or documentation style.
I like some of your logic. Which is very different from mine. Some of the lines took me a second to understand what you were thinking when you wrote it, but once I got it, it made total sense.

[1]https://www.python.org/dev/peps/pep-0020/
[2]In case a team-member does not contribute, please email simon@simula.no

Imports

You have used some different types of imports in the different scripts. I can find both *import ** and *import numpy as np*, which is not very consistent.

```
import *
```

Importing all in Python is generally frowned upon. It clutters the namespace, and the function/class/thing you are actually importing and using later in your code; we would not know from which package that function is imported from.

let's say in each of your integrate, numpy_integrate and numba_integrate modules you have called your function just "integrate", if you did import * on all, the pointer to integrate would be overwritten for each import.

Complicated/Cryptic

I don't find/see anything I would classify as overly complicated, but there are some cryptic and some shortened variable names here and there that could have just had the entire word as variable name or something descriptive. i.e. *l_f_a*

Documentation (Docstrings/Comments)

All functions should have a docstring on the line just below def. Example:

```
def integrate(f,a,b,N):
    """ integrates function f from a to b with N Ns, and returns the result """
    \# code here
    \# ...
    \# return result
```

Assignment 4.1

Add a review based on section 1.2. Do the tests have a meaningful name?

The tests have meaningful names. I'd like to see tests for numba and cython as well. I'd also like to see tests with N being either 0 or a negative value.

- The code is working as expected. Maybe add a list of Ns so we can easily test different N values, including N=0 and N equal to a negative number.
- I do not miss any docs/comments in the tests. The code documents itself.
- Some lines that are not super pythonic; split the variables to one per line, and write out tolerance instead of shortening it to tol.

```
        a = 0;  b = 1
        N1 = 10 ; N2 = 1000
        tol = 1e-10
```

I would probably have moved N1 and N2 into a list of Ns=[], and also tested for N zero and N negative. That would include tests for some errors we might expect to meet.

```
        a = 0
        b = 1
        Ns = [10, 1000]
        tolerance = 1e-10
```

```
import sys
sys.path.append("..")  # Fetches the directory the current subdirectory is a part of
from integrator import integrate
from numpy_integrator import integrate_numpy
```

Imports should be at the top of the file, and "sys.path.append("...") should be moved down.

- No overly complicated parts.
- I believe you are missing test of quadratic function i.e:

```
f = lambda x:x^2
```

ref assign 4.3

Assignment 4.2

Add a review based on section 1.2.

- The code woorks as expected for positive Ns. Add check on N for zero and negative numbers?
- I interpreted the assignment to calculate the left rectangles (ref the integration picture in the assignment at 4.0) also as written l [xi1, xi]
  Example:

```
for i in range(N):
    Sum += dx*f(a + i*dx)
```

```
for i in range(1, N+1):
    Sum += dx*f(a + i*dx)
```

- There are no docstrings
  Example:

```
    def integrate(f, a, b, N):
        """ integrates function f from a to b with N Ns. """
        # code...
        return sum
```

- It's all easy to read and not unnessesseray complicated.
- Variables in Python should generally be lower-case. ref PEP8[3] So "Sum" should be "sum".
- **"from numpy import *"**
  - "explicit is better than implicit"
  - "readability counts"
  - from your code you use

    `error = zeros(len(N))`

    so top of the file should be

    `from numpy import zeroes`

    or

    `import numpy as np`

    and later have used

    `error = np.zeros(len(N))`

    so when someone else are looking at your code, (or you come back to it in 3 months) we will instantly know that "zeroes" is a function imported from numpy, and not from any of the other libraries.

    `import *`

    also clutters the namespace.

At the plot error: Personally i might have made error = [] instead of error = np.zeroes() I feel like error=[] is more readable, but that might just be a personal preference.

Assignment 4.3

Add a review based on section 1.2. In addition, review the following assignment specific items:

- Is numpy being used effectively (e.i. vectorization where possible)?

I don't have anything to say about the logic, it looks OK, it's different from mine.
It doesn't *feel* pythonic, but I can't put my finger on it, so maybe it is?
dx should still be "dx = (b-a)/N"? That way the dx variable is self-explanatory? Now I feel like I would like a comment in there. Maybe just "# same as (b-a)/N" and maybe a reason to change it?

[3]https://gist.github.com/sloria/7001839#style

```python
from numpy import *
def integrate_numpy_midpoint(f, a, b, N):
        """ docstring """
        x = linspace(a, b, N+1)
        f_arr = zeros(N+1)
        dx = x[1]-x[0]
        f_arr[:] = f(x[:] +0.5*dx)
        Sum = sum(dx*f_arr[:-1])
        return Sum
```

You make two arrays here? x and f_arr, would it not be better to make x and do x[:] = f(x[:]) ? We also see here why you have chosen to name you variable sum for Sum. I would have done this:

```python
import numpy as np
def integrate_numpy_midpoint(f, a, b, N):
        """ docstring """
        x = linspace(a, b, N+1)
        dx = (b-a)/N
        x[:] = f(x[:] + (dx/2))
        sum = np.sum(dx*x[:-1])
        return sum
```

My understanding of 4.3 refers to 4.2 and 4.1 and you should have added a test test_integrate_quadratic_function (x = x**2) in addition to adding numpy_integrate to the tests.

Assignment 4.4

Add a review based on section 1.2.

- The code works as expected, but there's no tests for N=0 or N=neg value.
- Yes, we have a comment! :), and it's useful. Still missing docstrings.
- There's a function named "call", maybe call it at least "call_integrate" or something along those lines?
- Missing answer on "Can you think of any other advantages to using Numba instead of numpy?" ?

Assignment 4.5

Add a review based on section 1.2. If you are reviewing a INF3331 student, you can skip this review.

I did not do the cython version myself. But looking at it it looks like a nice cython implementation of your numpy solution.

It would be the same comments as for the numpy solution.

I like your reports :)

Assignment 4.6

Add a review based on section 1.2.

- Works as expected.No check for invalid Ns.
- Still no docstrings.
- comments:
  - `while error > tolerance`

    smart solution, i checked for that inside the loop. but you have set error to a random number, i would have done:

    `tolerance = 1e-10 for (..):          error = tolerance + 1`

    (or something) ...To specify that error should start with being bigger than *tolerance*, at first glance i wonder why it's set to *100000*. Maybe add a comment to show why.

    In the comparison file you have

    `error = abs(integral - 2)`

    where 2 is the "real answer" of the function, I would move *2* into a variable of it's own, so reading the operation is really clear on what it does.

    Midpoint implementation: Even though we all know that * comes before + i might have added a parenthesis around +(0.5*dx), for readability maybe +(dx/2), even though it is the same matematical operation, it's easier to understand? Maybe?

Assignment 4.7

Add a review based on section 1.2.

- Working as expected:
  - Can't pip install with the cython, something about path to visual studio.
  - I *think* the setup.py should have installed requirements? such as matplotlib (which in turn installs numpy.)
  - Everything else looks good.
- Still no docstrings, for a package; maybe we should have added a README with a short description of requirements, what it does and how to use it?
- Most of your code is compact, consise and self-explanatory, but I still miss some comments here and there.

Assignment 4.8

Add a review based on section 1.2.

Generating the functions is a neat little trick, but I think I would still write them out for readability. at least a comment on the last one that is not part of the generated.

variable names s, *l_f_e* should be descriptive. I would also use something else than "e" in the for loop? Maybe:

```
for func in func_list:
        f = eval("lambda x: %s" %func)
```

we are already used to using f=() for the name of the function variable from the rest of the assignments...

Summa summarum

All over, missing some docstrings, some tests, and comments, but good job and logic seems fine to me.