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

Jonas Lunde, jonassl, jonassl@student.matnat.uio.no
Håkon Holhjem, haakohol, haakohol@student.matnat.uio.no
Torkel Rogstad, torkelro, torkelro@student.matnat.uio.no

October 13, 2017

Review

The review was done using Python 3.5.3

Assignment 4.1

Your program gives me an AssertionError upon testing the constant function:

```
AssertionError: Test of NumPy integrator metohd failed, Diff: 1.998000e+00
```

This is due to a minor error in your numpy_integrate code, where you multiply by dx twice in cases where the $f(x)$ function returns a non-array element:

```
    E = dx*f(A)
return np.sum(E)*dx
```

After correcting this mistake, all the tests run smoothly with meaningful output. There is, however, a time-benchmark embedded in the test-functions, which serve no purpose, as the values are never used, and you have a seperate benchmarking function below.

Your implementation of looping over the functions with a zip-loop is nice and tidy.

I do, however miss some docstrings and comments in the code, even though it is rather self-explanatory, and well structured.

Assignment 4.2

The exercise technically specifies that you should use the *forward* values of the intervals as input to the function. This means using values in the inteval $[1,\ N]$ for calculating dx, and not in $[0,\ N-1]$, as you have.

The midpoint_integrate becomes correct, because you add an extra factor $\frac{dx}{2}$.

The time-benchmark worked fine, but took a while, even on my computer.

The plot is also good, but could do with some text on the axis.

Assignment 4.3

Your program doesn't run, because "N is not defined". Upon introducing an N, it runs fine, but there is no automization of the benchmarking between Python and Numpy, so I assume you did this manually. This makes me unable to confirm your results, but works fine.

The code is completely vectorized, which makes for an optimal decrease in runtime.

Assignment 4.4

The numba implementation works great. Your implementation of a wrapper function in order to pass a function as an argument (which numba doesn't usually accept) works very well.

You have also chosen to employ numba on the pure Python implementation, instead of the Numpy implementation. This has the clear advantage of not scaling memory usage with increasing N, which is a limitation of the Numpy approach.

You would see an even greater speedup with Numba by running with a larger N, as there is some overhead which reduces the time-difference for "small" N. You can also easily parallelize the code, by passing a "parallel=True" to "jit" and using the Numba-function "prange" instead of range.

Assignment 4.6

Your implementation of the midpoint method works great in every case.

The output of your integrator_comparison is very readable and informative. Your way of finding the lowest N that gives an error below 10^{-10} works find, but could be more robust (what if N=1e4 is already too large?), and is also somewhat slow.

The code also lacks some comments, as it isn't as obvious as alot of the other files.

Nice catch using the math modules sinus function for the non-vectorized cases, as numpys functions are slower for scalars.

Assignment 4.7

I was unable to locate any installation instructions or setup file for a module.

Assignment 4.8

The code gives rather accurate results with a reasonably sized N. There isn't really a whole lot to say about it, the code is rather self-explanatory.

Not sure why you are using the numpy implementation, when Numba is both faster and less memory-intensive, but it works fine for the chosen N.

There is also a little time-benchmark and informative prints. It seems you found an analytical solution for the functions, which is great.