1 MOD510 project feedback

Project 1:

Group number: 5

Abstract (4.5 / 5.0)

- Good abstract

Introduction (4.0 / 5.0)

- First paragraph is introductory in nature, and could be expanded upon a little. The rest is better placed in main part as a description.

Reflections (10.0 / 10.0)

- Excellent! Nice to read that you both have worked on the project individually and your constructive feedback.

Conclusion (9.0 / 10.0)

- Include your numerical findings for full marks.

Figures,tables (10.0 / 10.0)

- Good figures.

References (5.0 / 5.0)

- Conscious use of references!

CodingQuality (8.5 / 10.0)

- Good coding, and use of `*args`. Remember doc strings.

Analysis (19.5 / 20.0)

- Since the implementation was missing you are missing the analysis of the same question. Otherwise good work.

Implementation (23.1 / 25.0)

- In E2P1 you were asked to "Apply the same two function calls to the list. Explain what happens.". This was not done.

Bonus (0.0 / 5.0)

Overall points: 93.6 / 100.0

2 Expectations for projects 2, 3, and 4

- To get full marks, you should include the following (in addition to code and figures / tables): abstract, introduction, conclusion, personal reflections, and references.
- It is also very important to discuss your findings. Try to be as quantitative as you can. If you can interpret your results in light of known theory and/or empirical data, even better!
- In your handed-in Jupyter notebook, use your own words. Do not copy verbatim from the project text, or from other sources.

Final tip for writing a good report: Imagine that you are writing the report for a reader that knows nothing about the original assignment. How would you express yourself then?

3 General coding tips

Importing and plotting:

- Import all external dependencies (libraries, modules, etc.) at the top of your notebook, and do it only once.
- By typing `%matplotlib inline` at the top of your Jupyter notebook, you never have to invoke `plt.show()` to produce figures. This makes your code more compact, and it becomes easier to customize figures after their initial creation.

How can you write clearer code?:

- Try to reduce code repetition by defining smart functions and/or classes.
- Any single function should not do too many things, because that makes it harder to re-use the function.

  - Example: It is almost never a good idea to have a single function do both model calculations *and* plotting.

- It is very important to document any code that is not 'self-explanatory'.
- On the other hand, you should not comment things that are completely obvious from reading the code.
- Focus on improving code clarity rather than cluttering your code with comments (which quickly become obsolete as the code changes).
- Choose descriptive names for variables/functions/classes etc.
  - Examples of a "bad" function name: `function1` (what kind of function is it?)
- Explain how your code is to be used by writing docstrings (see below).
- The PEP style guide contains many useful tips for how to structure your code.

Docstrings:

- When defining your own functions, classes etc., it is considered good practice to include a docstring.
- The docstring is a special string literal placed right after, e.g. a function definition, to explain the purpose of the function and how to use it.
- It is recommended to use triple double quotes `"""` around docstrings.
- Add 'r' in front of the docstring if it contains a backslash (the 'r' stands for *raw string*)
- When calling Python's `help` function, the docstring will be printed.