Reflection Report on SynthEddy

Phil Du (Software)
Nikita Holyev (Theory)

1 Changes in Response to Feedback

Most changes made to the project were in response to feedback from reviewers are to improve clarity or consistency in the documents. For example, issue #9, issue #13, issue #22, and issue #61. This again proves the importance of having another pair of eyes to review the documents, as what may seem clear to the author may not be clear to the reader.

Revisions to the instance model, other than to improve notation and clarity, were generally suggested by Nikita Holyev after discussions, which then resulted in changes to how to program is structured.

Changes to requirements were mostly made in response to feedback from Dr. Smith.

GitHub issues were heavily leveraged to track these changes. Even for changes originated from myself, I often created an issue before start modifying the documentation or code. Besides keeping a record, this was very helpful as a reminder of what still need to be done.

1.1 SRS

Major changes:

- Add performance NFR issue #40 (which end up being too optimistic, need further changes)
- Move "realistic turbulent flow" Goal to NFR issue #42
- Remove two requirements (one moved to NFR) issue #52, issue #53
- Update TM, GD and IM to reflect the direction Nikita's research is going commit 86ae92d

1.2 Design and Design Documentation

Major changes:

- MIS: Rewrite the "Uses" section in each module due to initial misunderstanding issue #34

- MG: Change what classified as a "Software Decision Module", due to initial misunderstanding issue #37
- MIS: Improve access routing documentation at multiple places, e.g issue #35, issue #73
- Change what is in a "eddy profile" after discussion with Nikita, from quantity based to density based, so that the same eddy profile can be used on different sizes of fields issue #72

1.3 VnV Plan and Report

Major changes:

- Expand rational to explain test case to non-expert readers issue #28
- Specifying Expected Knowledge Level issue #29
- Rewrite verification plan sections according to feedback by Dr. Smith issue #57, issue #58

2 Design Iteration (LO11)

There was a mismatch of expectations initially regarding the scale of the problem. I was not aware of the number of eddies (10 million) and meshgrid point (1 billion) that Nikita was trying to work with. This led to me not giving enough considerations to performance and tried to implement the module "by the book", strictly following the modular design and instant model.

The performance of the initial implementation ended up being unable to realistically handle the real life scale of the problem. This led to a major redesign of the Flow Field and Eddy modules. Some modularity was sacrificed for performance, with more information be consolidated into the Flow Field module for faster vectorized operations.

The chunk approach was also not part of the original design or theory, but implemented to save memory and reduced unnecessary computation. This was inspired by the chunk design of Minecraft.

I do wonder how can we better document, or even better, anticipate such implementations. I had a feeling that, due to the scientific computing nature of this project, a lot of the focus was given on turning mathematical models into program. However, the real-world consideration of performance and "programming tricks" were not given enough attention.

3 Design Decisions (LO12)

Other than the module change and chunk approach discussed above, three other design decisions were made that are not related to performance or mathematical

model: the use of JSON file for eddy profiles and queries, inclusion of File I/O Module and Query Interface Module.

The JSON file was chosen for the expected use case of the program. With large meshgrid, the program will likely be submitted to run on a cluster. The JSON file allows the user to easily pre-define the eddy profiles and queries, instead of needing to input them every time the program is run. These files may also be generated by other programs.

The File I/O Module was included to handle the reading and writing of files needed by multiple modules. This module is used to handle various file formats in various subdirectories local to the program directory. Even though for each file format, there is a dedicated way to handle file I/O (JSON decode, NumPy load, etc.), the File I/O Module is used to abstract the file handling process so that the other modules do not need to know how to handle them and work with the file paths.

The Query Interface Module seems a bit redundant at this stage, as it merely passes information between the Main Control Module and the Flow Field Module during a query. But this leaves the possibility that, if the program is to be run on a server as a service, the Query Interface Module can be expanded to have different query methods, such as REST API, or even via a GUI.