Geosci. Model Dev. Discuss.,
https://doi.org/10.5194/gmd-2018-3-AC2, 2018

Geoscientific Model Development Discussions

Interactive comment on "Veros v0.1 – a Fast and Versatile Ocean Simulator in Pure Python" *by* Dion Häfner et al.

Dion Häfner et al.

mail@dionhaefner.de

Received and published: 1 July 2018

Dear referee,

thank you for your thoughtful review and the recommendation to accept our paper. We address the issues you raised point-by-point below. You will also find a `latexdiff` of the revised document as a supplement.

Kind regards,

Dion Häfner
on behalf of the authors

1 General Comments

> -Introduction,line 6: ...using the low level programming language fortran ... I think it should be mentioned that fortran is only one of the programming languages used in the ocean model community, not THE language. There are plenty of models that are also written in C or C++.

Re-worded these sentences to be a bit more general.

> -Introduction, line 8-10: ...violates a core principle of science: reproducibility ... I don't see how complexity violates reproducibility. Its the job of the programmer and the development community to do implementation step by by step and that new implemented features are carefully tested, but this inherits any kind of programming effort independent of the used language.

Complexity per se indeed does not automatically *violate* reproducibility. However, we live in a world where resources are finite, and especially so in academia. A more complex model code that is poorly abstracted *does* lead to a decrease in "testability", and since people's time is precious, this particular piece of code will be, on average, less well tested.

Changed the wording of this sentence ("violate" → "jeopardize").

> -Introduction, line 11-12: ...designed with the explicit goal to improve code structure and readability ... Any code in any language can be structured and commented that "anybody" can read and understand it, but only when the responsible programmer cares about. The difference in python to all other languages I know is, that the structure of the code (line intents, tabs, spaces ...) is a necessary part of the syntax of python, which forces the programmer to structure its code to a certain extend.

Our understanding of structure in a program goes far beyond whitespace. Python (such as many other high-level languages) actively advocates the usage of modularity, clear scoping rules, and other modern software engineering best practices. Furthermore, the community-wide coding standard PEP8[1] demands a consistent style of all Python projects, lowering the bar of entry for new collaborators.

Please also consider the following quote from our response to the first review comment:

"While it is in principle possible to write clear Fortran code with meaningful abstractions that may be just as readable as a high-level implementation, the reality is often different. Popular ocean models such as MOM [4] or POP2 [5] feature subroutines that are hundreds to thousands of lines long, and both models rely on more obscure Fortran features such as `COMMON` blocks, which makes it hard to keep track of variable scopes for inexperienced programmers. This is not necessarily due to flaws in Fortran's core design, but we do consider the established idiomatic style of a community to be tightly bound to the language used."

> -Introduction, line 16-17: ... a substantial amount of ... projects is devoted to understanding, writing, and debugging legacy Fortran code... Understanding writing and debugging is part of any kind of model programming effort, irrespective of the used language, that is a burden one always has to deal with. I think the big advantage of the high level programming language python is, that its first unless like MATLAB fully open source, so there are no nasty licensing issues to address for any package and second that the running of the code and the visualization of any kind of model variable can be done theoretically together. This would make it much easier, especially for beginners, to understand what is going on in the model and speed up any debugging work-flow considerably. Low level programming languages only allow limited output to the screen/log-file or need own complicated output routines to write out more complex variables which are visualized with

> something afterwards which makes it often time and resources consuming to find the origin of bugs.

The emphasis of this sentence is not on *understanding, writing, and debugging*, but on *substantial amount of the duration* and *legacy Fortran code*. The debugging workflow in a Python environment is very different from that in a Fortran project. Python code can be pulled apart dynamically, run interactively in a Jupyter notebook, and (as you correctly note) tightly integrated with visualization tools. On top of this, Python is one of the most popular programming languages in the world, which means it is much easier to get help. All of these factors contribute to a significantly more pleasant experience for everyone who is not a Fortran expert.

On a side note, there is a multitude of stronger reasons why one would choose Python over MATLAB than it being free software (some of which are outlined in Sect. 2.3). In my personal opinion, a project like Veros in MATLAB would be pointless, since most of Python's advantages don't apply.

> - 2.1 From Fortran to naive python, line 28: ...arbitrary indexing in Fortran ... I'm not sure what the author means here with the term "arbitrary indexing". Also indexing in Fortran is anything else than arbitrary.

This refers to the fact that the index range of an array can be chosen by the programmer in Fortran [1], while Python arrays always start at 0. Slightly reworded this remark to make that clearer.

> - 2.3.3 Multi-threaded I/O with Compression, line 27: ... ranging from Gigabytes to Petabytes... I haven't met yet any model application where single output files in the size of Petabyte where written. Did the author meant Terabyte ?

This sentence refers to “data sets” by which we mean the entire output of, say, a model run. In long-running high-resolution models, this easily reaches the Petabyte scale (e.g. the experiment outlined in [2]).

> - 2.3.3 Multi-threaded I/O with Compression, line 29-30: Since writing output becomes more and more to a critical bottleneck especially, for large model configurations, it would be nice if the author could describe a bit more in detail how writing the output is organized in VEROS especially with respect to the separated threads. How is pre- vented that the data are overwritten when the model runs further, while one thread is writing out ?, Are the output data duplicated for writing the output?, Does it affect the RAM demand of the model? ...

Good point. All data is copied in-memory to the output thread before continuing. This indeed increases memory consumption temporarily, which is why this feature can be disabled with a flag. In practice, we have not experienced issues with excessive memory usage.

Added a sentence to Sect. 2.3.3.

> -2.3.5 Modular Diagnostic Interface: How does VEROS structure the output, does it follow the CMIP protocol, one file for one variable or it combine several variables into a file?

Currently, we loosely follow the CF conventions (`http://cfconventions.org/`) in our netCDF output files. So far, most of the output format is inherited from pyOM2, but we plan to be strictly compliant with CF in the future. For all default diagnostics, we use one file per diagnostic (e.g., all snapshot variables are written to one file, and all temporal average variables to another). But we consider this a detail that might change soon, so we don’t want to get into too much detail in this paper.

> -4.1. Modified Geometry with flexible Resolution: The author should mention what he used as forcing to obtain these results

Indeed. Added a sentence to 4.2.

> It would be nice if the author could also make some statements about the memory (RAM) demand between VEROS and pyOM2, when running the same configuration. Are they the same?, Are there differences in the size of the model configuration that VEROS can handle compared to pyOM2...

Added a paragraph on memory consumption to Sect. 3.2.

2 Technical Comments

> - page1, line15: ...to further advance our ...

Replaced “further” by “advance” for clarity.

> - page3, line26 (same page8, line9): 2.1 From Fortran to naïve Python

Replaced the somewhat archaic spelling “naïve” with the more modern “naive”.

> - page9, line24: ...and the implementation of

Re-worded.

- page16, line21: ...(Chelton and coauthors et al. ,1998)
- page16, line25: ...(Chelton and coauthors et al. ,1998)

Bibliography and reference styles are as supplied by Copernicus, which we have no control over.

- page19, line 34: ... Last Glacial Meaximum ...

Corrected.

- page 21,line 4: ...PopularitYy...

We follow the official spelling of PYPL here, which is indeed “PopularitY”.

3 Other Changes

Consistent spelling of pyOM2 with lower-case “p”.

4 References

[1] https://www.python.org/dev/peps/pep-0008

[1] https://docs.oracle.com/cd/E19957-01/805-4940/z400091044d0/index.html

[2] Poulsen, Mads B., Markus Jochum, and Roman Nuterman. "Parameterized and resolved Southern Ocean eddy compensation." Ocean Modelling 124 (2018): 1-15.

Please also note the supplement to this comment:
https://www.geosci-model-dev-discuss.net/gmd-2018-3/gmd-2018-3-AC2-supplement.pdf

---

Interactive comment on Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-3, 2018.