Topical editor comments

Reviewer comments in black

Author's responses in blue

Dear Authors,

Thank you for having carefully addressed the comments from the two reviewers. The modified version of the manuscript is in good shape and could certainly be accepted for publication in GMD upon addressing the minor revision suggestions I list hereafter.

Best regards,

Ludovic Räss - Topical Editor

Thanks for the useful suggestions. We have adopted all suggested edits, and, in particular, have removed some of our potentially controversial comments about interpreted vs compiled languages. We will look into Julia in future work. Thanks again, Ludovic, for your kind support dealing with our paper.

Specific comments:

l.20: Why only listing Matlab and Python as interpreted languages? Did you try these two? If no, I would suggest to only keep interpreted languages without example.

Yes – we have coded up solutions to certain problems discussed in the paper in both Python and MATLAB. We have updated the model repo associated with this paper to include a MATLAB version of the model (https://github.com/amireson/openRE)

Also, as a general comment, SciPy is mostly compiled as most of SciPy's functions actually call compiled C or Fortran code

We agree.

l.23: "code is short and simple" -> concise?

Agreed – change implemented.

l.141-147: Did you look into and/or try Julia as language? Julia has an interesting DifferentialEquation.jl environment that exactly addresses solving PDEs using the method of lines. Could be worth checking out briefly and also mention it in here.

We will investigate this, but we have not had an opportunity to try this out yet. We cannot mention this in the current manuscript, without further exploration.

Fig.1: Are you using a staggered grid? Of so, maybe worth mentioning it referencing it. Staggering will actually make the scheme conservative.

We had originally used the term “block-centered grid” which is common, but we have now updated this to “cell-centered grid” following Bear & Cheng 2010, where this type of grid is clearly described. This is precisely equivalent to a staggered grid, but that terminology is not, to our knowledge, used in our field. We updated our terminology from “block” to “cell” throughout, and we added the note on line 170: “(note, this is equivalent to what is called a “staggered grid” in fluid dynamics)”.

Section 2.4.2: The statements reported here are somewhat inaccurate:

- "Interpreted languages are compiled on the fly, meaning every individual line of code is compiled at run-time" -> not true. Interpreted languages are interpreted, not compiled. Some functions maybe be written in C, Fortran and (pre-)compiled (e.g. vectorised operations in Matlab, packages in Python such as Numba) leading to better performance.

- "By contrast, compiled languages are more efficient because the compilation and running of the code are separated, and you typically only have to compile the code once before it can be run many times" -> not true. Compiled languages are more efficient because they are compiled. Meaning, the compiler can specialise the expression, make lots of optimisations (loop fusion, and much more) as no interpretation needs to happen any more upon compilation.

Responding here to these two points - it is debatable whether it is fair to characterise an interpreted language as compiling each line at runtime – but we do not wish to say anything controversial in our paper that will detract from the message. We also agree with your second point about compilation. Therefore, we accept the criticism, and we have reduced and simplified the text in question to read: “Interpreted languages are not pre-compiled and are hence slower to execute than compiled languages. A nice compromise between the simplicity of interpreted languages and efficiency of compiled languages is to use a just-in-time compiler.”

- "A nice compromise between the simplicity of interpreted languages and efficiency of compiled languages is to use a just-in-time compiler. In Python, the numba library is such a just-in-time compiler (Lam et al., 2015)." -> That's correct.

Glad we agree on this.

- "Numba compiles all the lines of the Python code once at the start of the runtime, and then all subsequent calls to the code run much faster." -> Numba compiles only the subset of the code or functions marked as such.

Agreed – we modified the text to read: "Numba compiles selected Python functions once at the start of the runtime, and then all subsequent calls to the code run much faster."

As FYI, Julia would compile the entire code. Maybe worth mentioning it and even checking it out.

Thank you – we will certainly do this in future.

Section 3: for criteria iv) and v), it would be good to report which hardware you were running on (CPU, single/multi-cores, RAM, etc...) and elaborate a little bit more on the concept of "simplicity" of the code.

To address this comment in the introductory paragraph of section 3 (line 351) we added "For the purposes of comparing efficiency (*iv*), all simulations were run on the same laptop computer and we report the runtimes as a measure of relative performance. For the purposes of comparing simplicity of the code (*v*) we use a very simple metric of lines of code, which is reported above in Section 2."

Fig.6: May be good to annotate separate sub-panels, and fix the overlapping y-thick labels for the lower subplots

We modified the units in the lower plot to correct the overlapping y-ticks. We added annotations to identify each subplot and refer to these in the revised caption. We also added annotations to Fig 7.