Geosci. Model Dev. Discuss.,
https://doi.org/10.5194/gmd-2018-260-EC1, 2018

Geoscientific
Model Development
Discussions
Open Access

Interactive comment on "The Matsuno baroclinic wave test case" *by* Ofer Shamir et al.

D.A. Ham (Editor)

david.ham@imperial.ac.uk

Received and published: 5 November 2018

I will not pass comment at this stage on the relevance or correctness of this test case, since the nominated reviewers are much more qualified than I am to assess this. However I will address the issue of the reference implementation which has been provided. The reference implementation is important in a test case manuscript, as a well-executed reference implementation will greatly increase the attractiveness of the test case to model developers. Regrettably, at this stage the reference implementation has a number of issues which make it less likely to be used.

Printer-friendly version

Discussion paper

1 Deployability

Individual source files in a paper supplement are not particularly easy for language users to employ. This is unfortunate because Python and Matlab (though not really Fortran) provide straightforward mechanisms for packaging files to make them easily installable and usable. In the case of Python, if the package files are then hosted on an online revision control system such as GitHub then users will be able to directly pip install them with a single command. Instructions on how to create a Python package are at https://packaging.python.org/tutorials/packaging-projects/ while Matlab toolboxes are at: http://www.mathworks.com/help/matlab/matlab_prog/create-and-share-custom-matlab-toolboxes.html.

2 Verification and unit testing

There is currently no testing of the code provided. There is little more frustrating when using a test case than to eventually discover that the bug was actually in the test case. Potential users will therefore want to see that there is a robust set of verification tests that demonstrate that the implementation is correct. In this case you have two routes which can both be used. The first is to check the output of your routines for parameter values where the output is easy to verify. The second is to cross-verify your code by calling all the implementations and checking that their output is the same. For this purpose, it is useful to note that the Matlab code might be callable from Python using https://pypi.org/project/oct2py/. It is possible to directly call Fortran from Python using f2py, or you could use the Fortran standard C interoperability features to present a public C interface (which would also be a nice user feature!) and call that using ctypes. There is some documentation on how to do this in the https://docs.scipy.org/doc/numpy-1.15.4/user/c-info.python-as-glue.html.

You should use one or more test frameworks (maybe one, maybe you need one for each language) to run the testing in a way that users will understand and which is easy for them to integrate into their own testing frameworks. For example you might use https://pytest.readthedocs.io/en/latest/. There are likewise suitable testing frameworks available for Matlab and Fortran (or, if you set up cross-verification, you could just use the Python framework for all languages).

3 Code interfaces and naming

As far as I can see, the public interface of the code consists primarily of the function `get_fields`, which seems reasonable given that the purpose of the code is to execute one particular test case. However there are a number of things about this interface which are unhelpful. First, the name `get_fields` is problematic. It's not going to be obvious to the user that that is the main function and even what it does. A more descriptive name would be useful, potentially without the `get_` prefix, which adds no information. In Python, the auxiliary routines which are not supposed to be visible outside the package should be prefixed with underscore to mark them as private.

In both Matlab and Python, the documentation for functions should be placed in the docstring at the start of the function, not at the start of the file. This change will enable the self-documentation features of these languages to work. It should also be borne in mind that users may see the code without the paper, so it would be a good idea if the docstrings referred back to the paper.

The code returns values on a regular latitude-longitude grid. This is not useful for many users. Even many structured grid models have curvilinear grids which are not axis-aligned, and this is before considering unstructured mesh models. The functions should instead evaluate the results at an arbitrary set of latitude-longitude points. The modest performance gain that may be available from the current structure is not worth the cost of having code that many users simply will not be able to use.

It is quite conceivable that the user will want to change g, Ω, A, or H_0 in order to run the test case for a model using scaled parameters. Currently in Fortran these are private, and in all languages this change would involve changing the global state of the module, which is an unsafe programming practice. It would be better if these parameters were optional arguments to the main function(s), with default values being those given currently.

The test case is only valid for certain values of input parameters. In addition to documenting these limitations, the code should check for illegal values and raise appropriate exceptions or return some error, depending on language.

4 Licence

There is currently no licence provided with the code. This might mean that the code falls under the same Creative Commons attribution licence as the paper, but since the code may be separated from the paper when used, it would be preferable to provide an explicit licence. Many authors would use the MIT License in this context, which basically says that users can do anything they like so long as they continue to acknowledge the authorship of the code. See https://opensource.org/licenses/MIT. In any event, an explicit licence should be chosen and included with the source files.

5 GitHub and archiving

The steps above will result in quite a lot more files in the implementation, along with a nontrivial directory structure. This could be attached in the supplement, but this has a number of disadvantages. First, there is no mechanism for future bug fixes. Next, direct access mechanisms such as Python's package installer pip will not see the supplement. Instead, the code could be posted on GitHub, which would fix the update and pip problems. GitHub is not a suitable archive location for the canonical version of the code in a manuscript, but GitHub and Zenodo together provide a straightforward way of archiving the repository with a DOI for inclusion in the manuscript. The instructions are at https://guides.github.com/activities/citable-code/. GitHub + Zenodo would be a better alternative to the supplement.

Interactive comment on Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-260, 2018.

Printer-friendly version

Discussion paper