Geosci. Model Dev. Discuss., 1, S122–S126, 2008
www.geosci-model-dev-discuss.net/1/S122/2008/

Geoscientific
Model Development
Discussions

Interactive comment on "qtcm 0.1.2: A Python Implementation of the Neelin-Zeng Quasi-Equilibrium Tropical Circulation model" *by* J. W.-B. Lin

J. Lin

jlin@northpark.edu

Received and published: 17 December 2008

Many thanks to the referee for his kind and generous comments! They are greatly appreciated, as are his constructive and helpful suggestions! Below I address his comments using his numbering system.

Specific comments:

#1: In the second para. in the Introduction, the following sentence is included (based upon changing an existing sentence):

> Finally, as a compiled language, Fortran is non-interactive and requires

> separate compiling and linking steps. This hinders informal small-scale testing, prevents users from interacting with the model at run time, and can result in a longer development cycle.

#2: I’ve added Fortran 95 and 2003 as explicit examples. I’m not sure which aspects of “old” the referee has in mind to point out, but my sense is that too much specificity there might result in a tangential discussion on whether Fortran is old or new. By focusing on the fact that few scientific programs take advantage of the new Fortran features, I hope to circumvent that debate.

#3: The new last sentence of para. 3 is now changed to:

> Some modern languages are also interpreted; in those languages, source code is directly executed at run time without separate compiling and linking steps, thereby enabling interactive debugging and execution.

#4: Instead of the parenthetical statement “described later,” I put in the parenthetical statement “key model variables and parameters are instances of this class.”

#5: Likewise as in #4, I make the parenthetical statement read “which defines model objects.”

#6: A new subsection discussing OO is added right after the first paragraph in section 3. All section 3 are incremented up by 1, as a result. The text of the new subsection is below:

> Because Python is an object-oriented language, the fundamental programming unit is not the subroutine, but rather is the “object.” In a procedural language, data and functions that operate on data are two separate entities. In an object-oriented language, these two entities are bound together

in a single construct, the object. Because of this framework, functions are automatically considered in context with the data they operate on, and vice versa. This lessens the risk of errors that occur when data is manipulated by functions that were never intended to be used on that kind of data.

Data bound to an object are called “attributes” of that object, and functions that operate on that data are called “methods” of that object. In Python, the attributes of an object are specified by a name that comes after a period at the end of the object name. Thus, `model.runname` refers to the `runname` attribute of the `model` object. Methods are similarly named; however, to call a method, a parameter list (even if empty) must be specified. Thus, `model.run_session()` calls the `run_session` method bound to the `model` object.

In general, Python objects consist of two types of attributes and methods: public and private. Public attributes and methods are accessible to the general user. Private attributes and methods, on the other hand, are designed to be accessed only by developers. In Python, private attributes and methods have names prepended by one or two underscores.

Objects are created from a “template” that defines the attributes and methods that go into that object. The template is known as a “class,” and individual objects that are derived from a class are called “instances” of that class. Creating an object that is an instance of a class is known as “instantiating” the object. In the example above, `model` is an instance of the `Qtcm` class, which defines the `runname` attribute and `run_session` method. There is no limit to the number of instances of a class, and all instances of a class have equal access to the attributes and methods defined by the class.

Python’s highest level of organization is the package, a library of related modules. Modules, in Python, are individual files that define related objects, functions, and variables, and thus a package is a directory of module files.

A single module can contain an unlimited number of objects, functions, and variables.

#7: See #6.

#8: The parenthetical statement “e.g., units, long name, etc.” is added after “metadata.”

#9: Unfortunately, there are a few `Qtcm` attributes that are not model parameters or field variables at the Python-level, so to say there are only two sorts of attributes may not be accurate. I tried to make this part more readable by creating a new paragraph, starting with these two sentences:

> All model parameters (e.g., time step, etc.) are attributes of `Qtcm` instances. Field variables, at the Python-level, are also `Qtcm` instance attributes.

#10: Yes, the instance state encoded in the snapshot includes the date of the model, and thus the model will start from the date specified in the snapshot. Earlier in the paragraph where this line occurs, I add the line “The snapshot includes the date of the model and prognostic variables like `T1`.”

#11: I added the following sentences:

> Recall that *nearly any* model variable or parameter can be set via input keyword parameters. Thus, `inputs1` and `inputs2` could be different in the number of days the model is integrated, whether the land scheme is on or off, the initial values of the prognostic variables, etc.

in the paragraph after the block of code in this section.

#12: In the text, I add an example of reordering the list so that the convection scheme is called after all the other physics schemes:

```
    tmp = model.runlists['atm_physics1'].pop(0)
    model.runlists['atm_physics1'].append(tmp)
```

#13: I was a little confused what this referred to; I assume it was to the use of the name "cloudroutine" in Fig. 9. If so, I changed it to "cloudscheme" to avoid the confusion between thinking this run list name was the same as a function.

Technical corrections:

#1: Changed.

#2: Changed.

Thanks again to the referee for all his help! In gratitude, I've added his name to the acknowledgments section.

Interactive comment on Geosci. Model Dev. Discuss., 1, 315, 2008.