Object-Oriented Python Climate Models, Analysis Tools, and Undergraduate Teaching and Research

Johnny Wei-Bing Lin and Calvin Smythe
Physics Department, North Park University, Chicago, Ill.

Why Python?

Historically, climate modeling and analysis has used compiled languages like Fortran. The result has been brittle and inaccessible code. Wrapping compiled models in Python helps overcome these pitfalls because of the following features of this multi-platform and open-source language:

Interpreted and highly modular: Compiling and linking are done automatically. Program execution is dynamic at run time.

Object-oriented: Data, metadata ("attributes"), and functions that operate on that data ("methods") are bound together as objects.

Very clear syntax: Python programs read like pseudo-code.

Python climate tools

A variety of Python tools exist for both general scientific computing and climate specific research. Packages include:

Climate Data Analysis Tools (CDAT): Produced by the Program for Climate Model Diagnosis and Intercomparison at Lawrence Livermore National Laboratory, CDAT provides a suite of tools for data management, climate statistics, regridding, etc. URL: http://cdat.sf.net.

Matplotlib: 2-D plotting library (line, contour, etc.). URL: http://matplotlib.sourceforge.net.

Scipy: A comprehensive suite of mathematical and engineering functions. URL: http://www.scipy.org.

Interactive modeling with `qtcm`

The `qtcm` package is a Python wrapping of the Neelin-Zeng (2000) Quasi-equilibrium Tropical Circulation Model, a primitive equation-based intermediate-level atmospheric model focused on simulating tropical climate dynamics that includes baroclinic instability, a simple land-surface model, and a CAPE-based convective parameterization.

Fig. 1 shows a screenshot of an interactive Python session running an instance of the `qtcm` tropical atmosphere model. During an interactive session you have access to all variables in the current scope.

The visualization was done interactively at runtime. The screenshot also shows how you can change model variable values with an assignment statement, and continue the model run by calling `run_session` again.

Execution control using runlists

Because Python is an interpreted language, subroutine execution order and content is not fixed during runtime. `qtcm` uses "runlists," lists of string names, to describe what subroutines are executed and in what order. Fig. 2 gives an example of this kind of list.

```
phys = [ '__qtcm.wrapcall.wmconvct',
         '__qtcm.wrapcall.wcloud',
         '__qtcm.wrapcall.wradsw',
         '__qtcm.wrapcall.wradlw',
         '__qtcm.wrapcall.wsflux' ]
```

Fig. 2. List of physics routines to execute during one atmospheric timestep. Changeable at runtime.

Python provides built-in methods to append, insert, and replace items at will. For instance, to add a shallow cloud routine `shallowcloud` after the existing cloud routine, type `phys.insert('shallowcloud', 2)`. All regular programming constructs (looping, conditionals, operations, etc.) can also be used to manipulate the runlist.

Simple enough for all users

Through using Python as a wrapping language, students and researchers can use models without dealing with compiling and linking, makefiles, environment variables, shell scripts, etc., but rather can create, analyze, and visualize model simulations interactively. The flexibility of Python objects enables the user to alter a model's configuration on the fly, permitting easier access to more regions of the model's solution space.

For more information: Please contact Johnny Lin at:
Email: jlin@northpark.edu
Personal home page: http://www.johnny-lin.com
Package `qtcm` home page: http://www.johnny-lin.com/py_pkgs/qtcm

Reference and acknowledgements

Neelin, J. D. and N. Zeng, 2000: A quasi-equilibrium tropical circulation model—formulation, *J. Atmos. Sci.*, 57 (11):1741–1766.

Thanks to David Neelin and Ning Zeng and the Climate Systems Interactions Group at UCLA for their encouragement and help. On the Python side, thanks to Alexis Zubrow, Christian Dieterich, Rodrigo Caballero, Michael Tobis, and Ray Pierrehumbert for steering me straight. Thanks to God for allowing the `qtcm` package to run sucessfully. Early versions of some of this work was carried out at the University of Chicago Climate Systems Center, funded by the National Science Foundation (NSF) Information Technology Research Program under grant ATM-0121028. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

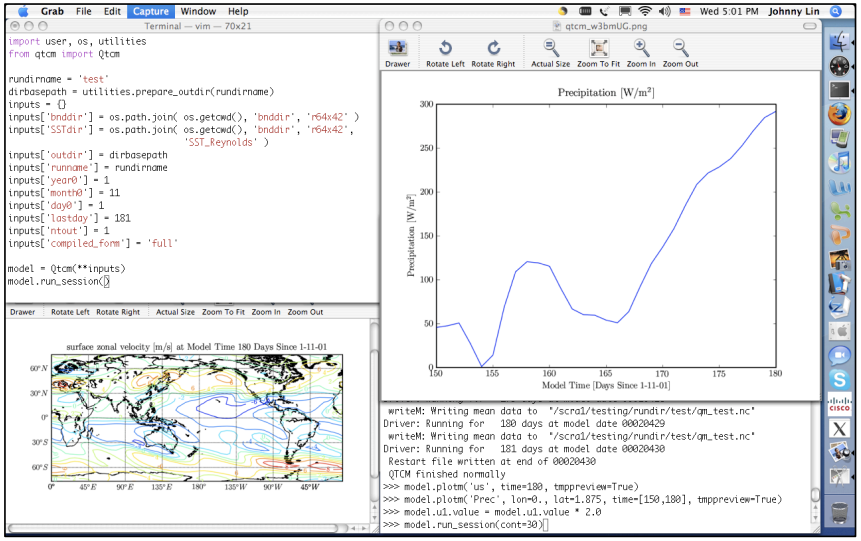

Fig. 1. Screenshot of an interactive integration of a `qtcm` model instance.

The upper-left window shows the code for initializing the model instance and running 180 days of simulation.

The lower-right window shows the run session. The first two lines in the window called the `plotm` methods to generate the two plots. The third line shows variable substitution for prognostic variable `u1` (doubling the existing value), and the fourth line will run the model for another 30 days when executed.

The 180 day model run took a little over a minute of wall-clock time on a 1.83 GHz Intel Core Duo with 1 GB 667 MHz DDR2 SDRAM running Mac OS X version 10.4.11. The horizontal grid for the model is 5.625 × 3.75 degrees longitude and latitude.