A Python Implementation of the Neelin-Zeng Quasi-Equilibrium Tropical Circulation Model and Implications for How Modeling Science is Done

Johnny Wei-Bing Lin
Physics Department, North Park University

February 22, 2012

Outline

Traditional climate programming: Development cycle and technologies

- Related to NWP but much development is research-driven.
- Code maintenance not a priority; documentation is sparse.
- Focus is on "just make it work."
- Many advances come from universities which lack the resources for substantial programming support.
- Written in Fortran which is fast but has limitations.
- Interfacing with operating system often unwieldy, through shell scripts, makefiles, etc.
- Parallelization requires the climate scientist to deal with processor and memory management (MPI) issues.

Traditional climate programming: Characteristics of codebase

- Brittle: Errors and collisions are common.
- Difficult for other users (even yourself a few months/years later!) to understand.
- Difficult to extend: Interfaces are poorly defined and structured between:
  - Submodels (e.g., between atmosphere and ocean)
  - Subroutines/procedures in a model
  - Models and the operating system
  - Models and the user (in terms of the user's thinking processes)
- Non-portable: Difficult for climate models and sub-models to talk to one another, sometimes even on the same platform.

Example of a traditional subroutine call

```
                  call ice_budget(sw_flux(i,j),lw_flux(i,j),
     &               sh_flux(i,j),lh_flux(i,j),ocean_flux(i,j),
     &               ocean_temp(i,j),ice_area(i,j),ice_thick(i,j),
     &               ice_temp(i,j),lead_temp(i,j),snow_thick(i,j),
     &               snow_fall(i,j),snow_temp(i,j),sfc_temp(i,j),
     &               sh_lead_flux(i,j),lh_lead_flux(i,j),
     &               u_air(i,j),v_air(i,j),ctr_ice(i,j),
     &               iceoc_temp(i,j),dhtop(i,j),dhbot(i,j))
```

- Fortran subroutine call to calculate the net energy budget in the sea-ice component from an early 2000's regional climate model of the Arctic (still in use).
- The argument list is long and unwieldy.
- The variables contain no metadata (and thus undetectable errors can easily propagate).
- Compiled languages are not interactive, which makes both programming as well as scientific use more difficult.

Attempts to improve climate modeling through modularization

- Modularization seeks to enable scientists to transparently swap climate model components in and out (e.g., ESMF).
- Modularization efforts are usually "monolingual": Fortran libraries/toolkits to abstract away memory management, etc.:
    - Inherent limitations of the language remain
    - Benefits of other languages are still unavailable
- Modularization usually only involves the models themselves. But, modeling as a *scientific methodology* involves more than just running the model:
    - Also involves hypothesis formulation, output analysis, etc.,
    - Needs to involve the use of other packages/languages and the operating system.
    - Modularization makes the models modular, but not necessarily the modeling *process.*

How a hybrid-language approach can improve climate modeling

- Implementing modern language features in an older “niche market” language (e.g., Fortran) is hard:
  - Backward compatibility is difficult
  - You’re essentially writing an entirely new language
  - Development expertise associated with the modern language is not generally transferrable
- An alternative: Leverage the strengths of multiple languages by creating hybrid-language models:
  - Get the benefits of modern object-orientation, interactivity, and an interpreted run-time environment
  - Get the speed/legacy code benefits of the older language

The Neelin-Zeng Quasi-Equilibrium Tropical Circulation Model (QTCM1)

Neelin & Zeng (1999) and Zeng et al. (1999)

- Intermediate-level atmospheric model.
- Vertical temperature and moisture profiles based upon convective quasi-equilibrium assumption.
- Betts & Miller (1986) moist convective adjustment scheme.
- Includes radiative-convective feedback package.
- Resolution 5.625 deg longitude, 3.75 deg latitude.
- Reasonable simulation of tropical climatology, and also includes Madden-Julian oscillation (MJO)-like variability. Used in MJO studies, ENSO studies, etc.
- Written in Fortran.

Overview of the Python qtcm package

- Software infrastructure:
  - Fortran: Numerics of QTCM1
  - Python: User-interface wrapper that manages variables, routine execution order, runs, and model instances.
  - Connectivity: Through the program f2py:
    - Almost automatically makes the Fortran routines and memory space available to Python.
    - You can set Fortran variables at the Python level, even at run time.
- Two main classes of objects:
  - `Field`: Key model variable and parameters.
  - `Qtcm`: A model instance.

A simple qtcm run

```
from qtcm import Qtcm
inputs = {}
inputs['runname'] = 'test'
inputs['landon'] = 0
inputs['year0'] = 1
inputs['month0'] = 11
inputs['day0'] = 1
inputs['lastday'] = 30
inputs['mrestart'] = 0
inputs['compiled_form'] = \
    'parts'
model = Qtcm(**inputs)
model.run_session()
```

- Configuration keywords:
  - Output filenames will contain the string given by `runname`.
  - Aquaplanet (set by `landon`).
  - Start from Nov 1, Year 1. Run for 30 days.
  - Start from a newly initialized model state.
- Run the model using the `run_session` method.
- `compiled_form` keyword chooses the model version that gives control down to the atmospheric timestep.

Run sessions and a continuation run in qtcm

```
inputs['year0'] = 1
inputs['month0'] = 11
inputs['day0'] = 1
inputs['lastday'] = 10
inputs['mrestart'] = 0
inputs['compiled_form'] = 'parts'

model = Qtcm(**inputs)
model.run_session()
model.u1.value = model.u1.value * 2.0
model.init_with_instance_state = True
model.run_session(cont=30)
```

- Make one run session.
- Double the value of `u1`, the baroclinic zonal wind.
- Make a continuation run for 30 more days.
- All can be controlled interactively at runtime.

Multiple qtcm model runs using a snapshot from a previous run session

```
model.run_session()
mysnap = model.snapshot

model1.sync_set_py_values_to_snapshot(snapshot=mysnap)
model2.sync_set_py_values_to_snapshot(snapshot=mysnap)
model1.run_session()
model2.run_session()
```

- Snapshots are dictionaries that act as restart files.
- `model1` and `model2` are separate instances of the `Qtcm` class and are truly independent (they share no variables or memory).

Runlists in qtcm make the model very modular

```
>>> model = Qtcm(compiled_form='parts')
>>> print model.runlists['qtcminit']
['__qtcm.wrapcall.wparinit', '__qtcm.wrapcall.wbndinit',
'varinit', {'__qtcm.wrapcall.wtimemanager': [1,]},
'atm_physics1']
```

- Run lists specify a series of Python or Fortran methods, functions, subroutines (or other run lists) to execute when the list is passed into a call of the `run_list` method.
- Routines in run lists are identified by strings. What routines the model executes are fully changeable at run time.
- Example shows a list with two Fortran subroutines without input parameters, a Python method without input parameters, a Fortran subroutine with an input parameter, and another run list.

qtcm performance is competitive with the Fortran-only QTCM1

System	Fortran-Only QTCM1	Python qtcm Package
Mac OS X: MacBook 1.83 GHz Intel Core Duo running Mac OS X 10.4.10	152.59	158.94
Ubuntu GNU/Linux: Dell PowerEdge 860 with 2.66 GHz Quad Core Intel Xeon processors (64 bit) running Ubuntu 8.04.1 LTS	43.73	47.45

Performance penalty of hybrid-languge model vs. the Fortran-only version of the model is 4–9%.

Wall-clock times (sec) for the average of three 365 day aquaplanet runs using climatological sea surface temperature as the lower boundary forcing (Lin 2008). All runs are executed as single threads.

Examples of qtcm uses: Conditionally explore parameter space I

Explore different values of mixed-layer depth (`ziml`) over a set of 30-day runs, as a function of maximum zonal wind associated with the first baroclinic mode (`u1`) magnitude, until you find a case where the maximum of `u1` is greater than 10 m/s.

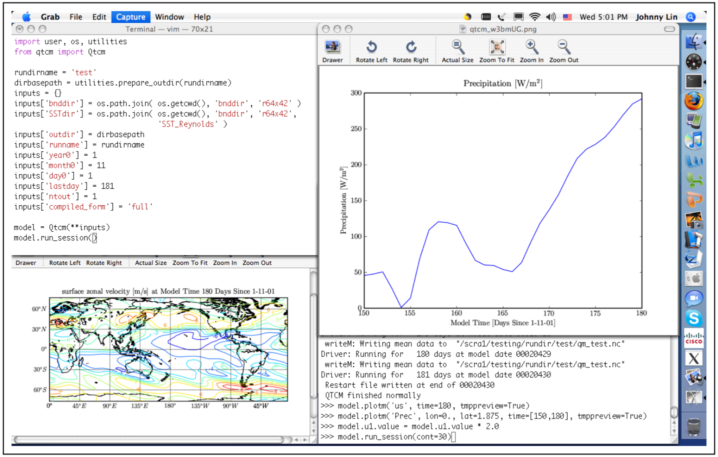
```
import os
import numpy as N
maxu1 = 0.0
while maxu1 < 10.0:
    iziml = 0.1 * maxu1
    iname = ziml- + str(iziml) + m
    ipath = os.path.join(proc, iname)
    os.makedirs(ipath)
    model = Qtcm(**inputs)
    try:
        model.sync_set_py_values_to_snapshot(snapshot=mysnapshot)
        model.init_with_instance_state = True
    except:
        model.init_with_instance_state = False
    model.ziml.value = iziml
    model.runname.value = iname
    model.outdir.value = ipath
    model.run_session()
    maxu1 = N.max(N.abs(model.u1.value))
    mysnapshot = model.snapshot
    del model
```

Examples of qtcm uses: Conditionally explore parameter space II

Using inheritance in Python to define and then explore the effects of multiple cloud physics schemes in multiple runs.

```
import os
class NewQtcm(Qtcm):
    def cloud0(self):
        [...]
    def cloud1(self):
        [...]
    def cloud2(self):
        [...]
    [...]
inputs[init_with_instance_state] = False
for i in xrange(10):
    iname = cloudscheme- + str(i)
    ipath = os.path.join(proc, iname)
    os.makedirs(ipath)
    model = NewQtcm(**inputs)
    model.runlists[atm_physics1][1] = cloud + str(i)
    model.runname.value = iname
    model.outdir.value = ipath
    model.run_session()
    del model
```

Examples of qtcm uses: Interactive modeling


Grab File Edit Capture Window Help
Wed 5:01 PM Johnny Lin
Terminal — vim — 70x21
qtcm_w3bmUG.png
Drawer Rotate Left Rotate Right Actual Size Zoom To Fit Zoom In Zoom Out
Precipitation [W/m²]
Model Time [Days Since 1-11-01]
surface zonal velocity [m/s] at Model Time 180 Days Since 1-11-01

```
import user, os, utilities
from qtcm import Qtcm

rundirname = 'test'
dirbasepath = utilities.prepare_outdir(rundirname)
inputs = {}
inputs['bnddir'] = os.path.join( os.getcwd(), 'bnddir', 'r64x42' )
inputs['SSTdir'] = os.path.join( os.getcwd(), 'bnddir', 'r64x42',
                                 'SST_Reynolds' )
inputs['outdir'] = dirbasepath
inputs['runname'] = rundirname
inputs['year0'] = 1
inputs['month0'] = 11
inputs['day0'] = 1
inputs['lastday'] = 181
inputs['ntout'] = 1
inputs['compiled_form'] = 'full'

model = Qtcm(**inputs)
model.run_session()
```

```
 writeM: Writing mean data to  "/scra1/testing/rundir/test/qm_test.nc"
Driver: Running for   180 days at model date 00020429
 writeM: Writing mean data to  "/scra1/testing/rundir/test/qm_test.nc"
Driver: Running for   181 days at model date 00020430
 Restart file written at end of 00020430
 QTCM finished normally
>>> model.plotm('us', time=180, tmppreview=True)
>>> model.plotm('Prec', lon=0., lat=1.875, time=[150,180], tmppreview=True)
>>> model.u1.value = model.u1.value * 2.0
>>> model.run_session(cont=30)
```

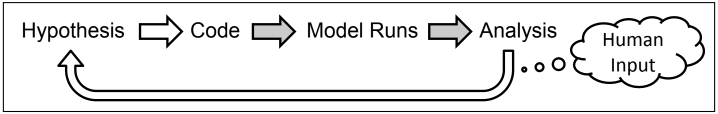
The graphs are created interactively by the user and can be used by the user to help set up another run session.

The effects of climate model programming structure on the modeling and analysis cycle

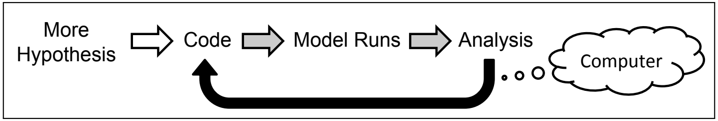
- Modeling has traditionally been a static exercise (i.e., set parameters, run, analyze output).
- The flexibility of changing i/o, data, variables, subroutine execution order, and the routines themselves at run time means modeling no longer needs to be static.
- Modeling is now more dynamic: The modeling study can adapt and change as the model runs.

Transforming the modeling and analysis cycle for climate modeling studies

Traditional analysis sequence used in modeling studies:



Transformed analysis sequence using qtcm-like tools:



Outlined arrows = mainly human input.
Gray-filled arrows = a mix of human and computer-controlled input.
Completely filled (black)-arrows = purely computer-controlled input.

Automating model output analysis makes more science possible

- Model output analysis can now automatically control future model runs. Try doing that with a kludge of shell scripts, pre-processors, Matlab scripts, etc.!
- Certain science questions that used to be difficult to access are now more possible to access:
  - For certain questions, code more closely matches user thought processes.
  - Automation enables more comprehensive searching of the solution space.
  - Each increase in code complexity can be more productive with a lower per line error rate.

Summary

- The traditional climate programming methodology works well in some ways but has inherent limits.
- A hybrid-language approach using Fortran for numerically intensive portions and Python for interface portions can leverage the benefits of each language to make up the deficiencies of the other.
- The Python qtcm package illustrates how a hybrid object-oriented approach can not only make climate modeling easier and more reliable but also enable researchers to investigate previously inaccessible (or difficult to access) questions.

For more information

- *Geosci. Model Dev.* paper on qtcm (many portions of this presentation copied/adapted from this paper):
  `http://www.geosci-model-dev.net/2/1/2009/`
- The qtcm Python package website:
  `http://www.johnny-lin.com/py_pkgs/qtcm/`
- The Neelin-Zeng QTCM1 website:
  `http://www.atmos.ucla.edu/~csi/QTCM/qtcm.html`
- Interested in learning Python or growing the atmospheric-oceanic sciences Python community? Come join PyAOS:
  `http://pyaos.johnny-lin.com/`