A simple object-oriented framework for making sea-ice models “plug-and-play”

Johnny Wei-Bing Lin
Computation Institute, University of Chicago

Introduction

Historically, sea-ice models have been developed incrementally and in compiled languages like Fortran. This has limited their modularity and robustness. The code is difficult to maintain and keep from becoming brittle.

Fig. 1 shows a snippet of Fortran code from an ice model. This snippet calls a subroutine that calculates the net energy budget. The argument list is long and unwieldy, the variables contain no metadata (and thus undetectable errors can easily propagate), and the grid is rigid, making it hard to couple the model to other climate models or components. Compiled languages such as Fortran are also not interactive.

Modern computer languages have structures that overcome many of these difficulties. Here we implement the Semtner (1976) “0-” and “n-layer” thermodynamic models in the open-source language Python in an object-oriented framework that results in a sea-ice model that is more “plug-and-play” than existing models. Because Python has a robust toolkit to interface with compiled code, the framework described here can be extended in a straightforward way to “wrap” existing Fortran sea-ice models (such as CSIM).

```
              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))
```

Fig. 1. A subroutine call from the sea-ice model in a regional climate model for the Arctic. Most climate scientists would consider the complexity of this code as nothing unusual.

Modular sea-ice models

Older procedural languages usually lack tools to manage the variable namespace. One result is that subroutine argument lists are “hard-wired” and lengthy. Modern languages avoid this through dictionaries and the creation of specialty data structures and classes to ensure the right variables are available and used when needed.

Fig. 2a shows code used to initialize the “0-layer” model (for clarity, two lines used to initialize forcing are not shown). All of the parameters, boundary conditions, and variables, along with their metadata, are stored in the `init_state` object. When the model is initialized (last line), only one argument is then needed. Note that the `init_state` object is *not* a Fortran common block. Because it also contains variable metadata, the object is verifiably self-consistent.

Fig. 2b shows initialization code for the “n-layer” Semtner model. The code is *identical* to 2a, except for the last line. What this illustrates is not that the initial conditions are the same, since they cannot be: the ice layer temperature variable in the `initialize_run`

(a)
```
init_state = initialize_run()
set = StateSet(init_state, delt=8.0*3600.0)
model = SimpleIce(set)
```

(b)
```
init_state = initialize_run()
set = StateSet(init_state, delt=8.0*3600.0)
model = Semtner(set)
```

Fig. 2. Initializing the (a) “0-layer” model and the (b) “n-layer” model.

function, for instance, has only one “layer” for the “0-layer” model but n layers for the “n-layer” model. What it does illustrate is that by binding variable metadata to the variable object itself, needed information (e.g. the grid) automatically propagates upwards to the model creation level. The object management tools also automatically ensure that only needed variables are used by the model. Thus, it is trivial to swap different models in and out.

Interactive modeling

Fig. 3 shows a screenshot of an interactive Python session running the “0-layer” Semtner (1976) model in a single-column mode. Because Python is an interpreted language, there is no separate compilation cycle, and during an interactive session you have access to all variables in the current scope. The upper left-hand window shows some source code and the lower right-hand window shows the results of a little less than two years of integration.

The visualization was done interactively at run-time. A user can also change parameter values at run-time and continue integration using those new values. This enables climate scientists to use the sea-ice models interactively.

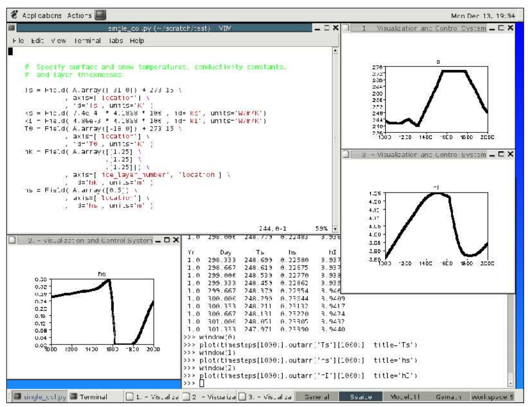

Fig. 3. Screenshot of an interactive integration of the “0-layer” model.

Grid independent models

Because object-oriented langauges allow you to carry within an object all the metadata needed to operate on the variable data also in that object, programs written in the object-oriented framework can readily be made grid independent.

Fig. 4 shows the code used to calculate change in snow thickness due to melting in the “n-layer” model. The first three terms on the right-hand side are masks ensuring only places with ice and snow and are above or equal to the melting point can have melting snow. The magnitude of the change is the timestep times the net heat flux out divided by the heat of fusion of snow.

```
hs_tend += hI_ne0 * is_melt * is_snow \
         * self.delt * (FA - Fs) / st0['qs']
```

Fig. 4. Change in snow thickness due to melting in the “n-layer” model.

What is unique is that this *code is the same regardless of domain type,* i.e. whether the domain is a single column, latitude bands, a latitude-longitude grid, etc.. It also does not matter whether the spatial discretization is regular (e.g. finite-difference) or irregular (e.g. a custom finite-element mesh). Again, since all grid information is attached to variable objects and operated on by attached methods, you can write code to do calculations without reference to unneeded domain information. In contrast, Fortran array size and dimensions are fixed. Changing from a finite-difference to finite-element grid would likely require changing the code: array declarations, array element indexing, etc.

Conclusions

Modern object-oriented computer languages enable us to create a sea-ice modeling framework that is modular, interactive, and grid independent. Such a framework can also be used to modularize other climate model components. Utilizing frameworks like these promises to enable climate scientists to focus less on programming and more on using the models for hypothesis testing and physical insight.

Reference and acknowledgements

Semtner, A. J. (1976): A model for the thermodynamic growth of sea ice in numerical investigations of climate. *J. Phys. Oceanog.*, **6**, 379-389.

This work benefited from conversations with Todd Arbetter, Rodrigo Caballero, Christian Dieterich, Marika Holland, Rob Jacob, Ray Pierrehumbert, Mike Steder, and Michael Tobis. In the final crazy weeks before this conference, God answered prayers that I would have results to present. This research was partially supported by National Science Foundation grant ATM-0121028. This poster solely represents the opinions of the author.

For further information

Please contact Johnny Lin at jlin@geosci.uchicago.edu. URLs:

http://www.johnny-lin.com/presn.html

http://climate.uchicago.edu/