Climate Data Analysis Software

Paul F. Dubois

Program for Climate Model Diagnosis and Intercomparison

Lawrence Livermore National Laboratory[1]

Abstract

Python is used as the basis for the construction of tools used by the climate modeling community. This paper briefly describes the Python-based software developed by the Program for Climate Modeling Diagnosis and Intercomparison at Lawrence Livermore National Laboratory, and briefly describes Pyfort, a tool used for making Python extensions with Fortran.

We also discuss our plans for expanding our software to become an "Earth Systems Grid", that is, a distributed environment for climate data analysis over the Next Generation Internet.

1.0 Introduction to climate modeling

Before we discuss the way in which we use Python to create software for the climate modeling community, we will briefly explain the nature of the work done by the climate modeling community.

1.1 Climate modeling and weather prediction

The weather is what you get; the climate is what you expect. While long-term weather prediction and short-term climate modeling are growing closer together these days, the basic distinction is that climate modelers try to predict the long-term average behavior of the climate and to understand the variability to be expected around that average.

One of the current issues for climate modeling is the well-known issue of "global warming"; that is, is there going to be an increase in the average temperature of the Earth over the next few hundred years caused by the increased amount of carbon dioxide and other so-called greenhouse gases in our atmosphere, and if so, how much will this increase be? Can we detect evidence for such an increase over the last few tens of years? While these are interesting and controversial topics, they are but one aspect of an effort to understand and model the geophysical environment.

The main tools of science in this area are observational data, experiments in understanding the basic atmospheric and geologic processes, and computational models which use the basic equations of physics together with geophysical data to simulate the behavior of the Earth over time. The output of such models can be studied to see what it predicts and validated by exploring the data to see how much verisimilitude it exhibits. For example, if a steady-state model predicts El Nino events at approximately the historically observed rate and intensities, this would be to its credit. Other criteria such as average cloudiness, rainfall, etc. can also be used to judge models

There are models that concentrate on the atmosphere, others that model the ocean, and so-called coupled models that model both together. Vegetation, ice and snow cover, and other factors are also considered by various models

1.2 The role of PCMDI

The Program for Climate Model Diagnosis and Intercomparison serves as a distributor for data produced by over thirty climate-modeling groups world wide, as well as for some observational data and other data derived from observations. PCMDI has been developing improved methods and tools for the diagnosis, validation and intercomparison of global climate models. PCMDI is a participant in the Department of Energy's Climate Change Prediction Program and has a number of projects such as the Atmospheric Model Intercomparison Project which collect model data and help understand that data.

2.0 PCMDI's Software Suite

PCMDI's software is a Python-based collection of modules for the analysis and visualization of climate model and observational data, GUI interfaces for user interaction, mathematical procedures for analysis, and tools for the exploration and standardization of data sets. To this we are adding components that will enable the distributed computing vision just discussed.

2.1 Architecture

Python is the key to our success. We have constructed our software as a collection of Python modules. Our users can use this software in many different ways: GUI-based tools for browsing and creating graphs, batch scripts for processing data sets, mathematical modules for computing derived quantities, etc. Python ties all these tools together and allows arbitrary combinations to be developed on the fly. For example, the user can invoke Python scripts which do intensive calculations or which use routines written in Fortran to help calculate quantities of interest to be plotted using the same tool used for making routine plots from models and observations.

Python has many attributes and facilities that combine to make this possible.

- The syntax is simple and easy to learn. This is crucial because the end users are not computer scientists. Their main experience is with Fortran, which shares with Python a generally similar expression syntax and array syntax, and the quality of being a language which can be learned completely in a short time.
- Python scales well with good namespace facilities that enable us to combine the use of many modules without problems arising.
- Python's object-oriented nature allows us to express naturally the concepts with which the scientist is dealing and to structure the software in a way that allows both expansion and modification without difficulty. Bertrand Meyer ([2]) calls this the "open-closed principle": the software has the property that it is open to further development but closed in the sense that current applications can count on a stable facility.
- Python's Numerical extension is important because it allows operation on very large, multi-dimensional arrays to be performed at near-compiled speed in Python. This greatly reduces the number of compiled extensions that are necessary and allows algorithm development and data exploration to be done mostly in the interpreter.
- Python's extensive international community and user-contributed library is a constant source of components that we can use. In return, Lawrence Livermore National Laboratory is supporting the Python Open Source effort by its membership in the Python Consortium and by releasing our own Python extensions and tools for use by others.

Key components of our software include:

- An object-oriented database facility that presents a uniform interface to the user to a wide variety of data formats used by climate scientists. Datasets can be described and manipulated logically as distinct from their physical representation. For example, the data for cloudiness in a particular model might be spread over a number of files representing different years, but the user need not be aware of that fact.
- Tools for mathematical processing of data. We have to have such tools that can correctly deal with missing values such as is common in observations or which may be desired in calculating effects while omitting adjacent data from, say, land areas. At first we did this by modifying the source for Numerical, but we are now able to do such calculations using a module written on top of the standard facility.

  Some of the things we do mathematically include statistical analysis, detection of anomalous data, regridding to new grids, and alteration of submitted data to match standards as to units, etc. Some of these calculations are computationally intensive and the performance of Python's numerical facilities and our ability to extend Python with Fortran is important.
- Visualization software, including a complex GUI interface called VCS.
- Tools to select datasets for examination by querying the metadata looking for datasets with the desired criteria.
- Tools to enable extension of Python. These are described further in the next section.
- Finally, we write tools that enable us to carry out our role as organizers, validators, and distributors of these large data sets.

2.2 The Next Generation Internet Project

The advent of significant broadband communications between climate research sites will enable researchers to more easily work together and enable distributed activities to examine datasets, select quantities for analysis, and perform calculations, with the ability to collectively view results and discuss them. PCMDI is participating as co-principal investigators in a Next Generation Internet project in which we begin to make such an environment available. While playing a leading role in this distributed computing environment, PCMDI is in cooperative activity with ANL, ORNL, LBNL, PNNL, LANL and NCAR. Making our software platform independent and based on current technology, PCMDI software is helping climate research scientist advance in scientific knowledge based on the physical ability of computers, databases, networks and associated computational software infrastructures.

Our recent demonstration of prototype software for the NGI project is an interesting case study of how Python enables rapid development.

We wanted to demonstrate using data located at different institutions to make a visualization. We wrote a data browser that allowed the user to select data either locally or via catalogs at different institutions, the catalogs being handled as an LDAP server. The user would choose data representing cloud fractions, surface temperature, terrain height at a certain time. The software would extract and transport the smallest necessary subset of the files required and then produce a three-dimensional visualization of the clouds and the terrain, with the terrain colored by temperature.

We had less than four months from the beginning of the project until the demo, and considerable time was spent understanding the technology available from our partners and elsewhere. In the end we utilized many different pieces of the Python software world:

- wxPython, a wrapper for wxWindows, a platform-independent open-source C++ library for GUI construction, was used to construct the GUI interface. It took only one man-month to go from completely ignorance of this package to successful deployment of a GUI with menus, notebook pages, pop-up dialogs, spin-counters, progress meters, etc.
- Necessary parts of ANL's Globus software for handling file transfers were wrapped into a Python module and plugged in to replace an FTP component in the already-working demo. No recompilation of our demo was needed since we accomplished this as a dynamic Python module. This was important in that we did not need to set up our ANL partners with a complete development environment.
- The newly-available Distutils let us write a setup and installation script for our partners, who had never seen Python and had no idea how to build an extension for it. Distutils is being developed at CNRI by Greg Ward as part of the distuils-sig effort to create easy methods for constructing and installing Python packages.
- The standard Python library supplied us with key components for the original FTP version, handling of URL's, communication with the LDAP servers, as well as the usual use of packages such as os, string, and sys.
- Vtk, a 3-D visualization package, was used to make the final picture. It had a Python interface that allowed us to prototype what we wanted, although bugs in the barely-developed Python interface eventually drove us to run it under tcl instead for this particular demo.

2.3 The scientists get in on the act

One of the frequently-observed benefits of this sort of computational steering, in which an interactive scripting language is used to perform scientific calculations, is that it enables the creativity of the scientific staff to be brought to bear in order to solve their own problems. This allows for the computer science staff to be much smaller than it would have to be if every request for enhancement or every need for a new algorithmic component had to be achieved by modification to an "official" program. With our approach, there is no bottleneck; users can do many things for themselves and directly enhance the software to solve their own problems.

The more computationally-adept members of the community create their own tools, including small GUI interfaces, based on our components. This allows in turn other people to run the software easily and accurately

3.0 Tools

Two of the tools developed for our work may be of interest to others. The newest one, Pyfort, is a tool for connecting Fortran to Python. CXX, described in a paper at the Seventh International Python Conference, is a tool for creating Python extensions using the C++ language rather than C. These tools and their documentation are available at the LLNL Python website, http://xfiles.llnl.gov. A link to this page is "Python at LLNL" on www.python.org.

3.1 Pyfort

Pyfort is a tool for connecting Fortran routines to Python. While connecting Python to Fortran routines is possible using such tools as SWIG ([3]), doing so requires considerable knowledge and a syntax unfamiliar to a Fortran-programming scientist. Since Fortran's 1990 standard included a syntax for describing the interfaces to Fortran routines, it seemed natural to build on that syntax as an input language to a Python / Fortran connection tool.

For example, here is the Pyfort input used to create a Python extension module "amodule" that connects Python to a routine f (n, x, y), where x is an array of length n that calculates results and stores them in an array y, also of length n.

```
module amodule
    subroutine f (n, x, y)
    integer n = size (x)
    real x(n)
    real, intent (out):: y (n)
    end
end
```

The language used to describe the routine f is nearly identical to the Fortran interface block that would be written to ensure type checking in Fortran 95. Here the declaration of the integer n lets the user indicate that n is to be calculated from the size of the array x, so that it doesn't have to be passed as a separate argument from Python. Use of this feature is optional, of course.

After running Pyfort and creating the extension module, a process that takes just a few minutes, a dynamically-loaded module is available for importing into Python. The user can then call the Fortran subroutine f from Python using a natural syntax:

```
y = f (x)
```

Pyfort is capable of generating a sample input file for the Distutils. The user simply has to fill in the appropriate details to build and install the package. We hope to capture the community's knowledge about different Fortran compilers and systems to complete this facility.

3.2 Recent developments in CXX

CXX is a package of header files and supporting run-time source that enables the construction of extensions to Python in C++. The idea behind CXX, as described in the paper presented last year [4], is twofold. First, the Python C API is wrapped into a series of classes such as Object, Dict, and Tuple. C++ exceptions are converted into Python exceptions and all intermediate objects are cleaned up automatically in such as case. As a result, reference counting errors are nearly eliminated, along with complicated cleanup logic and extensive testing of return values from the Python C API.

For example, the following CXX code creates a dictionary and adds two named (Python) integers to it:

```
Dict d;
d [“one”] = Int (1);
d [“two”] = Int (2);
```

The CXX class hierarchy for this portion is shown in Table 1.

TABLE 1. CXX Class Heirarchy

```
Object
    Type
    Module
    Integer
    Float
    Long
    Complex
    Char (Strings of length 1)
    SeqBase<T>
        Sequence (= SeqBase<Object>)
        String
        Tuple
        List
        Array (NumPy array)
    MapBase<T>
        Mapping (= MapBase<Object>)
        Dict
Exception
    StandardError
        IndexError
        RuntimeError
    ... (more classes corresponding to the Python
exception heirarchy).
```

In addition there are a number of functions defined at the global (namespace Py) level. These include the usual binary arithmetic operators and stream output operators.

The second part of CXX is more experimental; it is an attempt to make it easier to construct your own objects and extension modules, without having to construct odd-looking tables and use mysterious non-standard constructions such as the infamous "staticforward" declaration. Barry Scott has substantially revised this section of CXX with some assistance from Paul Dubois.

Scott's new extension facilities for creating Python modules are now quite satisfying. For example, here is the coding to create a Python extension module named "example" containing a method "sum" that adds its floating-point arguments:

```
class example_module : public
ExtensionModule<example_module>
{
public:
    example_module():
ExtensionModule<example_module> ("example" )
    {
        add_varargs_method ("sum", ex_sum,
            "sum (arglist) = sum of arguments");

        initialize ( "documentation for  module" );
    }

    virtual ~example_module () {}

private:
    Object ex_sum (const Tuple &a)
    {
        Float f (0.0);
        for( int i = 0; i < a.length (); i++ )
        {
            f = f + Float (a[i]);
        }
        return f;
    }
}
```

Note the support provided by the CXX machinery. For example, if one of the arguments to "sum" is not a Python floating-point number, an exception is thrown, any temporary objects such as f are cleaned up, and a Python exception results.

Now we add the initialization routine Python requires, which now must only construct a permanent instance of our module class:

```
void initexample()
{
    static example_module *example =
            new example_module;
}
```

This example has been simplified from the one in the CXX package for expository purposes. Potential users should consult the files in the Demo directory for further examples, including use of the new extension object facility.

4.0 References

1. This work was produced at the University of California, Lawrence Livermore National Laboratory (UC LLNL) under contract no. W-7405-ENG-48 (Contract 48) between the U.S. Department of Energy (DOE) and The Regents of the University of California (University) for the operation of UC LLNL. The views and opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.
2. Bertrand Meyer, "Object-Oriented Software Construction", 2nd Edition, Prentice-Hall, N.J., 1998.
3. Beazley, D.M. "SWIG and automated C/C++ scripting extensions". In Dr. Dobb's Journal, 282 (Feb. 1998),p. 30-36.
4. P. F. Dubois, "A facility for extending Python in C++", in *Proceedings of the Seventh Internation Python Conference*, Foretec Seminars, Reston,VA, 1998. pp. 61-68.