Report on OOPS C++ training

Mohamed JIDANE : DMN, Casablanca, Maroc/Morocco
Mentor : Claude FISCHER, CNRM/GMAP

January 2012

The OOPS project :

The data assimilation algorithms can generally be expressed mathematically by several generic equations involving a small number of abstract vectors and operators. These equations do not depend on details of a specific numerical prediction model, and can be applied to many different systems. However, implementations of most algorithms for data assimilation are very specific to the model and it is not possible to take an algorithm used in one model and apply it to another.

One of the main motivations for Object Oriented Prediction System (OOPS) is to provide a clean environment, and especially flexible framework for data assimilation algorithm that separates its specific implementation in any one model. Other motivations for OOPS are the code cleaning and re-factoring in the control level of the IFS System, in order to get rid of some present difficulties of the code (too big complexity, too many logical switches, burden of global variables and setup).

The Object Oriented Programming is an attractive technique for addressing these goals. OOPS is the related project in the frame of IFS (ECMWF). It is mainly written in C + +.

In this context, Météo France held a first C++ training between 16 and 20 January.

The aim of the course was to give an overview of C++ programming by solving several practical problems. The course also focused on the Object Oriented Programming concepts.

One of the main objectives when programming with objects is to organize programs more effectively. Objects are the key programming concept for implementing: encapsulation, abstraction, inheritance and polymorphism.

The one-week duration of the course was not sufficient enough to catch all the difficulties of a versatile but complicated language like C++.

The following week, I was involved in installing and running the OOPS code delivered for the technical review of June 2011.

Guidelines on how to build and run the OOPS toy system on Linux PC :

Above all, we need to get and install BOOST libraries :

1. Download `boost_1_48_0.tar.bz2`.

2. In the directory where you want to store BOOST, execute

   ```
   $ tar --bzip2 -xf /path/to/boost_1_48_0.tar.bz2
   ```

3. Most BOOST libraries are **header-only**: they consist *entirely of header files* containing templates and in-line functions, and require no separately compiled library binaries or special treatment when linking.

   If you want to use any of the separately compiled Boost libraries, you'll need to acquire library binaries.

   Issue the following commands in the shell :

   ```
   $ cd path/to/boost_1_48_0
   ```

   you'll probably want to at least use

   ```
   $ ./bootstrap.sh --prefix=path/to/installation/prefix
   ```

   Finally,

   ```
   $ ./b2 install
   ```

Now we can start building OOPS:

1. In the directory where you want to install OOPS, execute :

   ```
   $ tar -zxvf oops_review.tgz
   ```

2. Type the following command to ensure you use Linux architecture :

   ```
   $ export ARCH=linux
   ```

3. Specify the path to `boost/include` in the `config/linux.mk` file :

   ```
   BOOST_INC := -I $(HOME)/boost/include
   ```

4. Then, to build the executables, execute :

   ```
   $ make $model
   ```

   where `$model` determines which model you will be using. Currently, the options are "`l95`" for the Lorenz-95 model, or "`qg`" for the QG model.

To run the OOPS toy system, you will need to :

1. create a few directories (otherwise the run will fail) :

   ```
   $ mkdir -p out ../Data/FDB
   ```

2. There are three executables for each model. To run an assimilation, you will typically need to run the following four steps:

- Generate truth:

```
$ ./${model}_forecast.x ${model}/scripts/truth.xml
```

- Generate observations from the truth:

```
$ ./${model}_makeobs.x ${model}/scripts/makeobs4d.xml
```

- Perform a short forecast from the truth run, to generate the background:

```
$ ./${model}_forecast.x ${model}/scripts/forecast.xml
```

- Run incremental 4dVar:

```
$ ./${model}_4dvar.x ${model}/scripts/4dvar.xml
```

Data files are saved in the "Data" directory .

3. To test the toy system , execute the script `runtest.ksh` which will run several of the main configurations for the Lorenz-95 and the QG toy models and compare some of the key outputs with a control output stored in the `control.summary.txt` file . A proper control summary file should be obtained from a partner Center (having already run the toy tests) or from an independent local installation (which already was validated). For my installation on PC, I compared with a reference file kindly provided by Yannick Trémolet (ECMWF).

   Before running `runtest.ksh`, two small corrections should be done in the script : re-direct the output of `./qg_4dvar.x qg/scripts/4dsaddlepoint.xml` into directory `out`, and use `tkdiff` instead of `bbdiff`.

   The differences between outputs are just rounding differences.

4. Plotting section :

   Currently, plotting is available for the QG model only. There are two python scripts:

   - `qg/scripts/plotFields.py` (plots a QG model state)

   - `qg/scripts/plotDiffs.py` (plots the difference between two states)

   The path to python should be corrected in those two python scripts (adapt to your local path).