

A&OS 180/229 – Writing VMO history files

Spring, 2008 – Fovell

For visualization, we will employ a locally-developed package called VMO, or View Model Output. Development of VMO was begun in 1991 by Peter Dailey and Robert Fovell and is a Fortran-based interactive front-end to NCAR Graphics. VMO is scriptable and can be used to make basic contour plots, color contours, color shading, plot overlays and animations.

VMO will help us examine our model output with a minimum of fuss and distraction, as long as you use my code for creating VMO-compatible files. For use on our Synoptic Lab Linux workstations, subroutines, executable files and other information needed for this class will be stored in the directory `/home/fovell/AOS180/`. **You should copy files into your own home directory or scratch space** to use them, but also check back occasionally for updates.

VMO has also ported to Mac OS X; check with me for that version.

In this directory the following files should appear:

- `ModelingNotes2005.pdf`, the latest version of the AS 180 course notes in PDF format;
- `prefs.vmo`, the preferences file VMO requires and reads on startup;
- `vmo2008.pdf`, a VMO command reference document;
- `historynew180.f`, Fortran subroutine for creating VMO compatible history files. This subprogram is used to create a history output files called `history180F`;
- `readhist180F`, an executable which reads `history180F` files and makes VMO readable output;
- `historynew180.C`, a file which contains the C++ version of the history generator routine. This subprogram is used to create a history output file called `history180C`;
- `readhist180C`, an executable which reads `history180C` files and makes VMO readable output;

You may have to alter your environment to run VMO successfully. As of Spring, 2008, the following alterations have to be made for users of the Synoptic Lab Linux machines. **Please read and follow these directions carefully.** Edit your `.cshrc` file. If you have a `.tcshrc` file, edit that instead. You need to alter your `PATH` and `LD_LIBRARY_PATH` information.

(1) Back up your present file, using the `cp` command.

(2) Find the first reference to `PATH` in your file. Insert this as the next line

```
setenv PATH /home/fovell/vmointel:$PATH
```

(3) Find the first reference to `LD_LIBRARY_PATH`

```
setenv LD_LIBRARY_PATH /opt/intel/fce91/lib:$LD_LIBRARY_PATH
```

Variables VMO needs and expects

Please read this carefully and create or modify your code as instructed below. For either Fortran or C++, `historynew` requires certain constants and arrays be passed to the routine, including some additional 2D arrays you may not have created yet:

- `ihcnt`, a counter which is set to zero for the first call to `historynew` and unity for all subsequent calls;
- `nx`, `nz`, `dx`, `dz`, `time`, with values set within from your main program. `time` is the current model time, in **seconds**. For the first call to `historynew`, `time` should be set to 0;
- `tb`, `qb`, `ub`, `rhov`, `rhov`, `pi`, `vb`, all base state arrays, all dimensioned `nz`. The `vb` array is designed to hold the base state north-south wind; you can fill it with zeroes, but you need to create it;
- `u`, `w`, `th`, `pi`, 2D arrays from your model, dimensioned `nx` by `nz`;
- `qv`, `qc`, `qr`, `km`, `v`, additional 2D `nx` by `nz` arrays present for legacy reasons. You can use these to compute and write your own derived or prognosed fields (such as Richardson number, passive tracers, etc..), or just fill them with zeroes;

Fortran implementation

Steps to implementing history output by your Fortran program:

1. Create arrays `historynew` expects in your main program;
2. Open a new output file called `history180F` in your main program, before calling `historynew` for the first time. The file should be text, and use unit 70. The open statement I use employs status “unknown” which means that any previously existing `history180F` file in the working directory will be overwritten. That open statement is:

```
open(70,file='history180F',status='unknown',form='formatted')
```

3. For the **first** call `historynew`, call the routine with `ihcnt` set to 0, as in:

```
ihcnt = 0
call historynew(ihcnt, nx, nz, dx, dz, time,
1  tb, qb, ub, rhov, rhov, pi, vb,
2  u, w, th, pi,
3  qv, qc, qr, km, v)
```

4. Subsequent calls to `historynew` set `ihcnt` to 1.
5. Close your unit 70 file when you're ready to end the integration.

```
close(70)
```

C++ implementation

For the C++ version, I am presuming you have defined your prognostic variable arrays as globals (i.e., at the top of your program, before the `int main()` declaration. The function prototypes are:

```
// VMO prototypes
void historynew(int ihcnt, ofstream& fout, int nx, int nz,
               double dx, double dz, double time,
               double tb[nz], double qb[nz], double ub[nz], double rhou[nz],
               double rhov[nz], double pib[nz], double vb[nz], double u[nx][nz],
               double w[nx][nz], double th[nx][nz], double pi[nx][nz],
               double qv[nx][nz], double qc[nx][nz], double qr[nx][nz],
               double km[nx][nz], double v[nx][nz]);

void writehist(double a[nx][nz], ofstream& fout, int nx, int nz,
               int nxtodo, int nztodo, double multfac);
```

Since these arrays are defined as globals, they don't really need to be passed to `historynew`, but I felt writing it this way made it clear what's going on. The `writehist` routine is called by `historynew`; you don't need to mess with it.

Steps to implementing history output by your C++ program:

1. Create global variables `historynew` expects and add function prototypes to your program;
2. Open an output file called `history180C` in your main routine. The code I use is:

```
ofstream fout;
fout.open("history180C");
if (fout.fail())
{
    cout<<"Output file did not open."<<endl;
    exit(1);
}
```

3. For the **first** call `historynew`, call the routine with `ihcnt` set to 0, as in:

```
ihcnt = 0;
historynew(ihcnt, fout, nx, nz, dx, dz, timenow,
          tb, qb, ub, rhou, rhov, pib, vb,
          u, w, th, pi,
          qv, qc, qr, km, v);
```

4. Subsequent calls to `historynew` set `ihcnt` to 1.
5. Close your `fout` when you're ready to end the integration.

```
fout.close();
```

Postprocessing to VMO format

Postprocessing of history output:

1. Run either `readhist180F` or `readhist180C`, depending on whether you made your history file from Fortran or C++ programs. **The history files are plain text, by the way, and could be very large.** If you want to save them, rename and gzip (compress) them.
2. You'll see a greeting banner and then will be asked for a name for the VMO processed output. Consider a name like "task2" or similar (without the quotes). **Warning: this program cannot overwrite files.** That's a safety precaution, so if you want to reuse a name, you have to make sure there are no files with that name in the working directory.
3. The `readhist` routine then unpacks the history files into a binary format VMO can read. A file containing some common constants and grid setup information is created with the file extension `.cc`. Then, one additional file for each call to `historynew` is made; the file extension represents the model time the data represent.
4. If you want to get some practice with VMO, take a look at the VMO-ready files here:
`/home/fovell/AOS180/khi/`.

Using VMO

The VMO executable is called `vmo` and should be in your path if you set up your environment correctly. To launch the program, type `vmo` in the command line.

- On startup, an "NCAR Xgks" graphics window will appear and a color wheel will be drawn. After this finishes, click in this window to proceed.
- VMO will ask where your VMO binary output files reside. On three separate lines, as prompted, supply the directory, filename and the time you wish to access first. This need not be the starting time.
- The VMO prompt is `VMO>`. Commands are typed in after the prompt.
- Some common tasks (see `vmo2008.pdf` for much more information):

```
VMO> lt          list available times
VMO> pb=tp      plot basic contours of perturbation potential temperature
VMO> co=red, pg=tp  replace basic black contours with red
VMO> ci=0.4, pb=tp  plot using contour interval of 0.4
VMO> rb        enables red/blue contour line option (type again to disable)
VMO> ci=0.4, pg=tp  plot using red/blue color shading with interval of 0.4
VMO> wi=//0/1.5  crop window to between 0 and 1.5 km in vertical
```

VMO> **ti=480** change time to 480 sec (it must exist)
VMO> **cr=-1.5/1.5** limits contour range to min, max specified
VMO> **lw=2** doubles contour line width (thickness)
VMO> **m2** plot two plots per page; **sp** returns to single plotting

- To quit VMO, type **quit**
- VMO notes
 - By default, each plot appears in a new window. To overlay plots, type the **sa** (save) command. To free up the window – returning to the default behavior – type **fr**
 - For basic contour plots, setting the contour interval is optional. However, the contour interval must be set before doing **pg** color shading plots.
 - The **pc** command is not working. Use **pg** instead.
 - For contour intervals, enter an interval like "0.4" as "0.4". VMO is unhappy if you neglect the leading zero.
 - You cannot print from VMO. However, **every plot you create during your session is saved** in a file called **gmeta**. These images can be viewed, printed or manipulated into animations later.