**Octopus tutorial**

This tutorial intends on helping you using Octopus, a Lagrangian particle release experiment written by Jinbo Wang. The documentation for it is available at :

<https://github.com/jinbow/Octopus>

**Tutorial :**

1. Download the zip file at <https://github.com/jinbow/Octopus>
2. The directory contained in the zip file is called Octopus-master, put this directory somewhere convenient for you to work from (Desktop, an Octopus folder, etc).
3. Create 4 directories in Octopus-master : run, input, output, grid.
4. These 3 directories along with src and scripts are the ones you will be using for all of your experiments :
   * src : for “source”, contains all of the Octopus source code which is written in Fortran 90. This is the directory from which you can open and look into the different subroutines used by Octopus. It is also the directory in which you will compile Octopus before executing it.
   * scripts : contains different scripts written in python needed to initialize the particles of each experiment or process the output of Octopus.
   * run : contains the executable 0.particle, used to execute Octopus once you will have compiled it in src and copied in run, and anything to prepare your run.
   * input : contains your input files (velocities etc).
   * output : contains your output files (position of particles at each time step etc).
   * grid : contains the files of the grid of your input (hFacC.data, XC.data, XG.data, etc)
5. Move data.nml and data.nml.explained from src to run.
6. In src, change the size of your domain in size.h : Nx, Ny, Nz, Nrecs.
7. In src, edit cpp\_options.h to include or exclude features by defining them (#define) or undefining them (#undef). Make sure isArgo is turned off (#undef isArgo)
8. In src, compile the code using “make”. You will get an executable file named O.particle. Copy this file to your run directory.
9. Prepare the initialization file using scripts/init\_parti\_xyz.py. You will have to decide on the initial positions of the particles, convert the latitude, longitude, depth positions to i, j, k indices and save in a binary file in /run. Then run : python init\_parti\_xyz.py
10. Go to scripts folder and modify pth\_data\_out (the input folder) and pth\_data\_in (the grid folder) in gen\_data.py, as well as nz, ny, nx which are the size of your domain. Then run : python gen\_data.py to generate the binary files. After running gen\_data.py, you should get a list of binary files in your pth\_data\_out folder, including reflect\_x.bin and reflect\_y.bin and z\_to\_k\_lookup\_table.bin and k\_to\_z\_lookup\_table.bin.
11. Set parameters in the namelist file run/data.nml. The parameters are explained line by line in run/data.nml.explained.
12. In the run/ folder, run the model ./O.particle. Outputs are saved in the folder output\_dir (could be the folder output you created earlier) specified in data.nml.
13. In scripts, run : python glue\_opt\_data.py to “glue” all the output files in one file.
14. In scripts, run : python p\_xy.py to convert i, j and k indices to longitude, latitude and depth. You know have the lon, lat and dep vectors which correspond to the positions of the particles at each time step. Write up a script to plot them. You can use the Python package Basemap to plot maps with specific projections.

**Tips :**

* If you change domain make sure to change size.h and nz, ny, nx in gen\_data.py (run gen\_data.py again).
* If you change something in cpp\_options.h or size.h (or anything located in the src folder), make sure to re-compile by doing make clean then make and copy O.particle to your run folder. **DON’T FORGET TO COPY !**
* If you change the init\_parti\_xyz.py run it again to have new initial positions of particles.
* Each run, make sure to change the casename and the output\_dir in data.nml to not overwrite your previous runs.
* Make sure that the casenames and folder in glue\_opt\_data.py match the casename and ouput\_dir of data.nml.
* Make sure that the fn and folder in p\_xy.py match the casename and ouput\_dir of data.nml.
* When you write a new ouput\_dir in data.nml, no need to mkdir it before hand, Octopus will do it for you.
* I have uploaded my examples of init\_parti\_xyz.py, glue\_opt\_data.py and p\_xy.py on Google drive so you can get an idea of what they look like for me. (<https://drive.google.com/drive/folders/0B-bFFEL3qwXfc3ZQRXFEWlRMbEk?usp=sharing> )
* Remember to change npts (number of particles) in init\_parti\_xyz.py (thanks to scale in my example), data.nml, glue\_opt\_data.py and p\_xy.py.
* Remember to change NPP (number of releases/ensembles) in data.nml and glue\_opt\_data.py. The output files when NPP is greater than 1 will be one file for each time step you are saving at and for each NPP. For example if NPP=2, dt\_case=86400s and let’s say you are running Octopus for 10 days, saving every day, then you will get 10 output files for the 1st release and 9 output files for the 2nd release. Once you run glued\_opt.py you will get two files, one for each release. **The total number of particles is NPP\*npts !**
* You can save the output text of each run in a text file by doing : ./O.particle > output.txt. This will write everything in output.txt instead of in the terminal.
* When runs start taking a very long time, use the command screen in the terminal, this will open a Screen session, you can then write the Octopus run command (./O.particle) and quit this Screen session by doing **Ctrl-A + d**. Resume it later by typing screen -dr.
* Use symbolic links to put all the files you need in input and grid.
* In cpp\_options.h start by undefining everything

**Example of** data.nml **:**

Octopus is here initialized for a 10 000 particle run (Npts), advected for 5 years (360 days \* 5 = 155520000 sec, tstart to tend), released in one ensemble/one go (NPP=1), advected every 12 hours (dt), the output is saved every 10 days (saveFreq), the file names are written in days (DumpClock divides the time step of output being saved which is in seconds and writes this number in the file name).

&PARAM

pickup=0,

casename='10\_4PARTI\_0001',

path2uvw='../input/',

path2grid='../grid/',

output\_dir='../output/',

fn\_UVEL='UVEL\_ForEmmaV0.bin',

fn\_VVEL='VVEL\_ForEmmaV0.bin',

fn\_WVEL='WVEL\_ForEmmaV0.bin',

fn\_THETA='THETA\_ForEmmaV0.bin',

fn\_SALT='SALT\_ForEmmaV0.bin',

fn\_GAMMA='',

fn\_MLD='',

fn\_PHIHYD='',

fn\_parti\_init='parti\_init.bin',

target\_density=-1,

vel\_stationary=.False.,

Npts=10000,

dt\_reinit=-1,

dt\_mld=0.,

dt=43200.,

tstart=0.,

tend=155520000.,

NPP=1,

dt\_case=86400.,

saveFreq=864000.,

diagFreq=0.,

pickupFreq=2592000.,

Khdiff=25.0,

Kvdiff=1e-5,

DumpClock=86400.,

/

*Tutorial written by Emma Bent on June 23, 2018*

*Last updated on July 11, 2018*

*For more information or questions, please email me at* [*emjbent@gmail.com*](mailto:emjbent@gmail.com)