February 14, 2014

PyAverager

A package used for computing averages from climate model output.

Authors: Sheri Mickelson, Kevin Paul, and John Dennis
Version: 0.1.0

Comments and feedback: mickelso@ucar.edu

What the PyAverager Can Do

The PyAverager can create most of the needed climatology files needed by the AMWG, OMWG, Land, and Ice CESM diagnostic packages (the full list of averages is defined within the 'Specification' section). These averages can be computed with, where applicable. It is also able to compute averages from previously generated averages (such as season averages). The PyAverager requires filenames to adhere to the CESM naming conventions and currently can only operate on monthly time-slice and time-series files.

What to Expect from Upcoming Releases

- Will be able to operate on other model data with different naming conventions.
- Will be able to operate on data with time periods other than monthly.
- Will contain a unit testing suite.
- Provide command line support.
- New features: zonal averaging, averaging across ensembles, calculate standard deviation.

Dependencies

The PyAverager depends on the PyNIO and mpi4py packages to be installed on your system. PyNIO is needed for NetCDF file I/O. mpi4py is needed for the parallel communication, though it is possible to run the PyAverager in serial mode without mpi4py.

If you are not running on a CESM supported machine, installation information can be found at:
PyNIO: https://www.pyngl.ucar.edu/Download/
Mpi4py: http://mpi4py.scipy.org/
(git clone https://bitbucket.org/mpi4py/mpi4py.git)

CESM Supported Machine Information (steps to take to get these libraries into your path):

Yellowstone:
Add the following to the top of your bsub script or execute them on the command line and then type 'module save' to always keep them in your environment:
module load python
module load all-python-libs

Goldbach:
Currently installing modules

tukey.alcf.anl.gov:
Edit your $HOME/.soft.tukey file
Add (before @default):
    @PyNIO-1.4.1-anaconda
    @PyNGL-1.4.0-anaconda
    +mpich2-1.4.1p1
    +anaconda
Execute 'resoft'

edison.nersc.gov:
Follow the Yellowstone instructions, but load:
module load python_base/2.7.3
module load mpi4py/1.3.1
module load numpy
Add /global/u1/m/mickelso/python_libs/lib/python2.7/site-packages/PyNIO/ to your PYTHONPATH env variable

Building and Installing the PyAverager

- Check out the source code:
  svn co https://subversion.ucar.edu/asap/pyAverager/tags/v0.1.0

- Change into the top-level source code directory and run the Python setup
  - $ cd PyAverager
  - $ python setup.py install

- You can run the Python setup with the following options:
  - –-prefix=/path/to/install/to
  - –-user (will install the averager into your $HOME/.local directory

- Make sure the install location is added to your $PYTHONPATH
  - You can also type python to get the interactive terminal and then type
    import pyaverager
    import specification
    You will get an error if either is not in your path

- To install documentation, run:
  - $ doxygen Doxyfile
    The documentation will be created in the apidocs directory.

Running the Examples

Running the examples on Yellowstone:
(For other machines, you will need to create a queue submission script similar to examples/ runAvg_mpi.csh)

- $ cd examples
- Open runAvg_mpi.csh for editing
- Set the correct project number to run under
- Select an example to run (control_*.py)
- Edit the control_*.py script you would like to run
  (See the 'Specifier' section below for more details on editing the control script)
- Run
  - $ bsub < runAvg_mpi.csh

Specifier

The PyAverager is a python library that is referenced from another python script. In order to run the PyAverager, you need to specify parameters so the average knows what types of averages to compute, input/output locations, and any averaging options you would like to add. The 'example' directory contains several 'control_.py' files that you can use as templates. You can copy one of these scripts and modify the top section to fit your data.

CESM naming conventions that the PyAverager follows:

Slice: $CASE.$comp.$stream.$year-$month.nc

Series: $CASE.$comp.$stream.$var.$year1$month1-$year2$month2.nc

The table below lists the types of averages the PyAverager can compute.

Average Option	Description	Output Name	Can be Weighted?	Can Be Created As a Dependency?
ya	Yearly Average	$CASE.$YEAR.nc	Yes	No
tavg	Ocn average across years	tavg.$Year1-$Year2.nc	No	Yes
ann	Annual Average	$CASE_ANN_climo.nc	Yes	Yes
djf	Winter Average	$CASE_DJF_climo.nc	Yes	Yes
mam	Spring Average	$CASE_MAM_climo.nc	Yes	Yes
jja	Summer Average	$CASE_JJA_climo.nc	Yes	Yes
son	Fall Average	$CASE_SON_climo.nc	Yes	Yes
jan	January Average	$CASE_01_climo.nc	Yes	Yes
feb	February Avg	$CASE_02_climo.nc	Yes	Yes
mar	March Average	$CASE_03_climo.nc	Yes	Yes
apr	April Average	$CASE_04_climo.nc	Yes	Yes
may	May Average	$CASE_05_climo.nc	Yes	Yes
jun	June Average	$CASE_06_climo.nc	Yes	Yes
jul	July Average	$CASE_07_climo.nc	Yes	Yes
aug	August Average	$CASE_08_climo.nc	Yes	Yes
sep	Sept Average	$CASE_09_climo.nc	Yes	Yes
oct	Oct Average	$CASE_10_climo.nc	Yes	Yes
nov	Nov Average	$CASE_11_climo.nc	Yes	Yes
dec	Dec Average	$CASE_12_climo.nc	Yes	Yes
mavg	Concat of all monthly averages	mavg.$Year1-$Year2.nc	No	Yes
jfm	Ice Winter Avg	$CASE_jfm_climo.nc	Not Now	No
fm	Ice Feb & Mar Avg	$CASE_fm_climo.nc	Not Now	No
amj	Ice Spring Avg	$CASE_amj_climo.nc	Not Now	No
jas	Ice Summer Avg	$CASE_jas_climo.nc	Not Now	No
ond	Ice Fall Avg	$CASE_ond_climo.nc	Not Now	No
on	Ice Oct & Nov Avg	$CASE_on_climo.nc	Not Now	No

See examples/control.py for how to set all available options to send to the create_specifier function.

Variables that must be passed to the specification.create_specifier class:

- in_directory: directory where the input data is located
- out_directory: directory where the output will be produced
- prefix: the case name, plus component name (ie. b40.20th.track1.1deg.006.cam2.h0)
- file_pattern: currently not functional, any string will work

Optional variables that can be passed to the specification.create_specifier class:

- ncformat: either 'netcdf4c' (netcdf4 compressed (lev=1)), 'netcdf4' (netcdf classic), and 'netcdf' (netcdf3 classic) DEFAULT = 'netcdf4c'
- hist_type: either 'slice' or 'series' DEFAULT = 'slice'
- avg_list: a list of averages to compute DEFAULT = Empty List
  Format: ['ya:1850','mavg:1850:1890'] ya is the only average to take one year. All other averages expect a start year and end year separated by a colon.
- weighted: Boolean to weight averages (when available) DEFAULT = False
- split: Are the files split between lat coordinates (used in cice series files) DEFAULT = False
- split_files: strings differentiating the different pieces DEFAULT = 'null'
- split_orig_size: list of lat/lon names and their original full size DEFAULT = 'null'
- varlist: ['a','list','of','vars','to','avg'] DEFAULT = Full list

Can be created as a dependency? option:
The averages that are listed in the above table as being able to create averages as dependencies have the ability to use previously calculated averages to calculate a new average. To use this option, append 'dep_' in front of the average name (ie, 'dep_jja'). Without 'dep_', a jja average would loop over all June, July, and August values within the year ranges and create an average. With 'dep_', the PyAverager will create and output a June average, July average, and August average. Then it will open these average files and average these values to create the 'jja' average file. In most cases, it is faster to run with 'dep_', but it should be pointed out that the answers between using and not using the 'dep_' option will differ due to order of operation.

PyAverager Error Codes

errors 1-19: average list errors
1: Listed average is not in the know average list
2: Average cannot be created with depndencies
3: Average must list only one year
4: Average must have a start year and an end year
5: Date ranges are inconsistent and cannot run this average with dependencies

errors 20-39: input file problems
20: Cannot find the file (triggered in three different checks points)
21: Missing files to calculate DJF. You need either the previous December or the January and February from last year+1

22: Time series files are split, but the dates between them are not contiguous (triggered in two different checks points)
23: A date was found within two different time series files. Not sure which to use.