cmip5datafinder Documentation

Valeriu Predoi

Jul 12, 2017

Summary

- **Name:** `cmip5datafinder.py`
- **Type:** Python script
- **Language:** Python 2.7.13 (Python 2.7+)
- **Synopsis:** Script that searches for data locally (on a mounted disk specified by `--datasource`) and optionally also on valid ESGF nodes (using `--synda`). It builds cache files using the results of the search.
- **Needed python modules:** `sys, os, shutil, getopt, re, string, errno, numpy, subprocess, datetime, time`
- **Developer and contact:** Valeriu Predoi, University of Reading - (valeriu.predoi@ncas.ac.uk)

Introduction and examples

`cmip5datafinder.py` is a versatile Python tool that allows building cache files when run on a computer with access to a mounted disk where CMIP5 data is stored under the DRS path system and(or if synda is used) with access to a mounted /sdt/data synda data storage unit and a working synda executable. See below for explanation on synda. It requires minimal input from the user, with most of its functionality coded in as options. Below you can find the command-line options that can be used with the script, see Table 1.

The script uses either a parameter file that contains file parameters on each line or user-specified command-line file parameters.

The script is available to download () and comes prepackaged with an example paramater file (`example.txt`) to illustrate its functionality and so that the user can run the examples listed below.

Example 1: parameter file: workflow description

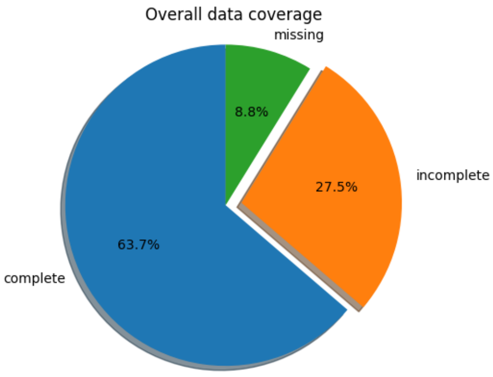
(without synda: command:)

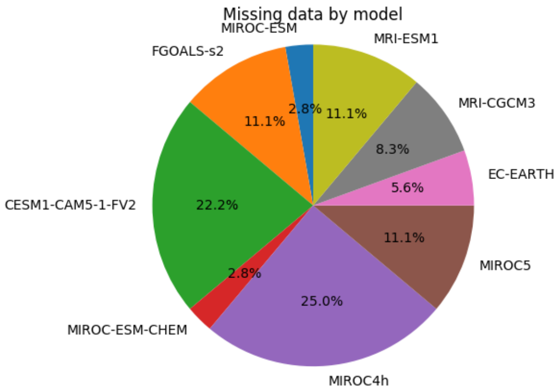
```
python cmip5datafinder.py -p example.txt --verbose --datasource badc
```

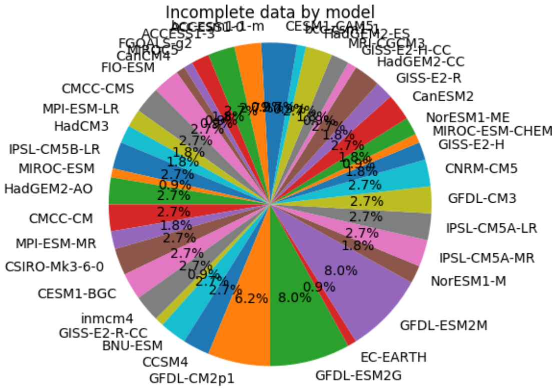
`-p, --params-file <file>`	Namelist file (xml) or text file (txt) or any other input file [REQUIRED] e.g. for xml: –params-file ESMValTool/nml/namelist_myTest.xml e.g. for text: –params-file example.txt e.g. for yaml: –params-file example.yml This option is REQUIRED if –user-input (command line) is NOT present
`-h, --help`	Display this message and exit
`--user-input`	Flag for user defined CMIP file and variables parameters (to be input at command line with –fileparams for each parameter) This option is REQUIRED if –params-file is not present
`--datasource`	Name of local data source (example: badc). Available datasources: badc [to add more here, depending where running the code][REQUIRED]
`--synda`	Flag to call synda operations. If not passed, local datasources will be used ONLY
`--download`	Flag to allow download missing data via synda
`--dryrun`	Flag to pass if no download is wanted. Don't pass this if downloads are neeeded! If –dryrun in arguments, all cache files will be written as normal but with NOT-YET-INSTALLED flag per file
`--fileparams`	If –user-input is used, this serial option passes one data file argument at a time If –user-input is used, this serial option is REQUIRED e.g. –fileparams CMIP5 –fileparams MPI-ESM-LR –fileparams Amon –fileparams historical –fileparams r1i1p1 –fileparams 1910 –fileparams 1919
`--uservars`	If –user-input is used, this serial option passes one variable argument at a time If –user-input is used, this serial option is REQUIRED e.g. –uservars tro3
`--verbose`	Flag to show in-code detailed messages

Table 1: `cmip5datafinder.py` command-line options.

The code will read each line in `example.txt`; each line represents a set of file parameters that are parsed (e.g. CMIP5 MPI-ESM-LR Amon historical r1i1p1 1900 1982 tro3); the code will look for relevant files only in the datasource `badc`, cache the files and output a cache file called `cache_example.txt-badc` and a directory called `cache_files_badc`. `cache_example.txt-badc` contains rows with four elements: filedescriptor (or header), level of completeness (`complete` or `incomplete` or `missing`), completeness percentage (a %.2f float) and the available file list. `cache_files_badc` will contain cache files (cache_cmip5 and missing_cache_cmip5) and images describing the file population by model. An example set of images below:







(with synda: command:)

```
python cmip5datafinder.py -p example.txt --synda --download --dryrun --verbose
--datasource badc
```

This case builds up on the previous case: the same search is performed within the `badc` datasource, but the **missing-only** filedescriptors are searched on ESGF nodes via *synda* (see https://github.com/Prodiguer/synda). Synda has the capability to complete the incomplete or missing filedescriptors, depending what ESGF servers are listed in its configuration file (we will not discuss this here since that is a system admin issue). Synda will first search in the locally-available locally-mounted `/sdt/data` disk, and cache whatever is needed and found there, and if (and only if) `--download` option is specified, synda will go on to search over the remote ESGF servers if files are still needed. `--download --dryrun` means that synda will search remote ESGF nodes but will not physically download anything found (it will, however, cache it as if it was downloaded but with the field NOT-YET-DOWNLOADED in the cache files for clarity). Synda-only files are stored in caches labelled `_synda_`.

Example 2: command-line args: workflow description

(with command line args) `python cmip5datafinder.py --user-input --fileparams CMIP5 --fileparams bcc-csm1-1 --fileparams --fileparams Amon --fileparams historical --fileparams r1i1p1 --fileparams 1982 --fileparams 2014 --uservars clt --uservars tro3 --uservars pr --datasource badc --verbose` Same as above only that the user specifies directly what file they need.

Easy-peasy!

Main functions

`cmip5datafinder.`**`cache_merge`**(*file1*, *file2*, *finalFile*)

Function that takes two cache files and merges them into a single one. Caution – note the order: file1 = local datasource cache file2 = local synda cache

:0 This function deals with different input date formats e.g. time1 = 198204 or time1 = 19820422 or time1 = 198204220511 etc More formats can be coded in at this stage. Returns year 1 and year 2

`cmip5datafinder.`**`date_handling`**(*time1*, *time2*) `cmip5datafinder.`

Function that generates the final user-friendly single cache file; this can easily be used in various analyses; file legend: Database | data_status | Percent complete | available_data ———————————— CMIP5_MIROC5_Amon_historical_r1i1p1_2003_2010_hus (complete,incomplete or missing) file$_{list}$

cmip5datafinder.**find_local_files**(*model*, *out1*, *dirname1*, *mfile*, *latest_dir*)

Function that performs local search for files using ‘find’ The depth is as high as possible so that find is fast. model: CMIP5 MPI-ESM-LR Amon amip r1i1p1 mfile: stderr dump file (cache_err.out) - need to capture instances of either Permission denied or non-existent dirs; latest_dir: latest version directory e.g. /latest/ on badc (see above for details)

cmip5datafinder.**fix_duplicate_entries**(*outfile*)

simple fast function to eliminate duplicate entries from a cache file

cmip5datafinder.**get_drs**(*dir1*, *sdir*, *ic*, *model*, *latest_dir*)

Function that returns DRS. dir1: root directory - /badc/cmip5/data/cmip5/output1/ sdir: subdirectory (institution) - MPI-M ic: experiment - MPI-ESM-LR model: CMIP5 MPI-ESM-LR Amon amip r1i1p1 latest_dir: on badc is /latest/ - this is known in advance and is dependant on where the code is run.

cmip5datafinder.**get_overlap**(*tt*, *my1*, *my2*)

function that returns the amount of overlap between needed data and available data Returns a fractional float li: list of years from data (1-dim, even number of elements) my1,my2: required model years

cmip5datafinder.**lsladir**(*dirname*)

Calling this function once so we save time; called in root dirname. It is needed for generalization and not hardcoding the institutions.

cmip5datafinder.**plotter**(*cachefile*, *saveDir*)

simple pie chart plotting function

cmip5datafinder.**print_final_stats**(*sfile*)

print some final stats To understand the output, by filedescriptor we mean any file indicator of form e.g. CMIP5_MIROC5_Amon_historical_r1i1p1_2003_2010_hus that is fully determined by its parameters; there could be multiple .nc files covering a single filedescriptor, alas there could be just one.

cmip5datafinder.**print_stats**(*outfile1*, *outfile2*)

small function to print some stats at the end

cmip5datafinder.**synda_check_dll**()

Easy checker on current downloads

cmip5datafinder.**synda_dll**(*searchoutput*, *varname*, *year1_model*, *year2_model*, *header*, *D*, *outfile*, *outfile2*, *download=False*, *dryrunOn=False*, *verbose=False*)

This function takes the standard search output from synda and parses it to see if/what files need to be downloaded

The searchoutput argument is a string and is of the form e.g.

new 221.2 MB cmip5.output1.MPI-M.MPI-ESM-LR.historical.mon.atmos.Amon.r1i1p1.v20120315.tro3_Amon_MPI-ESM-LR_historical_r1i1p1_195001-195912.nc done 132.7 MB cmip5.output1.MPI-M.MPI-ESM-LR.historical.mon.atmos.Amon.r1i1p1.v20120315.tro3_Amon_MPI-ESM-LR_historical_r1i1p1_200001-200512.nc new 221.2 MB cmip5.output1.MPI-M.MPI-ESM-LR.historical.mon.atmos.Amon.r1i1p1.v20120315.tro3_Amon_MPI-ESM-LR_historical_r1i1p1_185001-185912.nc

ie typical synda file search output. This gets parsed in and analyzed against the required model file characterstics and files that comply can be downloaded via synda install. It also takes the year1_model and year2_model, for time checks. It also takes the variable name and the name of a cache file outfile that will be written to disk. dryrunOn is the switch from a physical download to just polling the esgf node without any download.

varname: variable D: incomplete filedescriptors: the dictionary that contains the files that are already available locally year1_model, year2_model: needed filedescriptor year1 and 2 header: unique filedescriptor indicator e.g. CMIP5_CNRM-CM5_Amon_historical_r1i1p1_2003_2010_hus outfile: cache file outfile2: missing cache file download: download (either dryrun or for reals) flag

cmip5datafinder.**synda_search**(*model_data*, *varname*)

This function performs the search for files in synda-standard paths It takes exactly two arguments: - a model data string of type e.g. 'CMIP5 MPI-ESM-LR Amon amip r1i1p1' - a variable name as string e.g. 'tro3' It performs the search for files associated with these parameters and returns ALL available files. (command example: synda search -f CMIP5 MPI-ESM-LR Amon amip r1i1p1 tro3)

`cmip5datafinder.`**`time_handling`**(*year1*, *year1_model*, *year2*, *year2_model*)

This function is responsible for finding the correct files for the needed timespan:

year1 - the start year in files year1_model - the needed start year of data year2 - the last year in files year2_model - the needed last year of data WARNINGS: we reduce our analysis only to years

`cmip5datafinder.`**`which_synda`**(*synda*)

This function returns the path to the synda exec or aborts the whole program if synda needs to be used but its executable is not found.

`cmip5datafinder.`**`write_cache_direct`**(*params_file*, *ldir*, *rdir*, *outfile*, *outfile2*, *errfile*, *ld*, *verbose=False*)

Function that does direct parsing of available datasource files and establishes the paths to the needed files; makes use of find_local_files() File versioning is controlled by finding the ld = e.g. /latest/ dir in the badc datasource, this may differ on other clusters and should be correctly hardcoded in the code!

`cmip5datafinder.`**`write_cache_via_synda`**(*searchoutput*, *varname*, *year1_model*, *year2_model*, *header*, *outfile*, *outfile2*)

WARNING1: this function is SLOW WARNING2: this function is not currently used ———————————————— This function takes the standard search output from synda (synda_search()) and parses it to see if/what files exist locally

The searchoutput argument is a string and is of the form e.g.

new 221.2 MB cmip5.output1.MPI-M.MPI-ESM-LR.historical.mon.atmos.Amon.r1i1p1.v20120315.tro3_Amon_MPI-ESM-LR_historical_r1i1p1_195001-195912.nc done 132.7 MB cmip5.output1.MPI-M.MPI-ESM-LR.historical.mon.atmos.Amon.r1i1p1.v20120315.tro3_Amon_MPI-ESM-LR_historical_r1i1p1_200001-200512.nc new 221.2 MB cmip5.output1.MPI-M.MPI-ESM-LR.historical.mon.atmos.Amon.r1i1p1.v20120315.tro3_Amon_MPI-ESM-LR_historical_r1i1p1_185001-185912.nc

ie typical synda file search output. This gets parsed in and analyzed against the required model file characterstics and files that comply and exist locally are stored in a cache file for data reading. It also takes the year1_model and year2_model, for time checks. It also takes the variable name and the name of a cache file outfile that will be written to disk.