Imperial College London

Department of Earth Science and Engineering

MSc in Applied Computational Science and Engineering

Independent Research Project

Final Report

EZclim2.0 - A software package for Climate Modelling diagnostics

by

Yusen Zhou

yz219@imperial.ac.uk

GitHub login: acse-yz219

Supervisors:

Dr. Yves Plancherel

August 2020

GitHub repository: https://github.com/acse-2019/irp-acse-yz219

Abstract

Modeling on the coupled climate system helps us to understand the multi-scale dynamic interactions between natural and social systems that affect climate [1]. To examine model outputs expediently and efficiently, Adanna Akwataghibe from ACSE-2018 established EZclim, a new Python-3-developed software package. This software unites the reading, pre-processing and calculating process into one program for a diagnostics of climate modelling. EZclim is primarily concentrated upon establishing diagnostics' functionality as a consequence of time limitations. In other aspects, it has many flaws as well. It runs only on a local setting; hence the model output should be downloaded to the local machine at all times. It allots much duration and disk space for file downloading and it only accepts particularly small types of output to small enough to evade difficulties on storage memory. These problems have paved the way for the development of EZclim2.0.

Compare to EZclim1.0, EZclim2.0 was adapted to be ported to the high-performance computing (HPC) platform called JASMIN. EZclim2.0 can take advantage of its computational power to accelerate its calculation speed and read data from its data storage directly without downloading the data into the local machine. Besides, EZclim2.0 encapsulate the command-line style software package NCO into python code to preprocessing the data. This enables an attainment of necessary formats for various model outputs. To circumvent memory limitation issue, EZclim2.0 is now able to chunk the origin large size dataset into many small pieces. User submit these small pieces to Jasmin working server so there will not be a memory problem. EZclim2.0 also parallel some parts of the EZclim1.0 code to speed up the software, this ultimately gave the software a 15% performance improvement.

Introduction

1.1 Background

Climate modelling is similar with weather forecasting but focuses on changes spanning decades rather than hours [13]. The results of climate models promote the development of climate science and help scientists in understanding how human activities affect the Earth's climate [2][15][16]. A coupled climate system model is one such model that describes the multiple components of a climate system and the interactions amongst them [3]. Questions pertaining to coupled climate system modelling are usually large-scale and complex; moreover, they are hard enough to be resolved by a single team, scientific agency or nation. Thus, the World Climate Research Programme (WCRP) formed a working group of Coupled Modelling (WGCM) and began its Coupled Model Intercomparison Project (CMIP) [2].

Since 1995, the CMIP, which defines experiment protocols, forcing and outputs, has encouraged multiple international modelling teams worldwide to contribute to climate modelling experiments [4]. The outputs can be retrieved from the Centre of Environmental Data Analysis (CEDA), which hosts over 13 petabytes of atmospheric and earth observation data [5]. The CEDA also provides the JASMIN service, a unique data-intensive supercomputer for environmental science. JASMIN supports the data analysis requirements of the UK and Europe's climate and earth system modelling communities [5]. Through JASMIN, scientists can access curated data in the CEDA Archive and utilise the power of the supercomputer to accelerate research, but of course this also requires that analysis software must be written to accommodate the JASMIN environment.

The CMIP goes through a new iteration every five or six years. With the CMIP undergoing its sixth phase, over 70 models have participated in the model group, and most of them are newly added ones [1]. Compared to the previous stages, CMIP6 has some notable new features. For one, it has opted to use Swath/Curvilinear or Unstructured grids instead of rectangular ones. Swath-like data (hereafter, "SLD") are defined by non-rectangular and/or time-varying spatial grids, in which one or more coordinates are multi-dimensional [6]. This is a big challenge to EZclim. Use of non-rectangular grids complicates model intercomparing exercises since software must now be written to interpret the grid specific of each model or requires a regridding step. The following figures illustrate the differences of these grids.

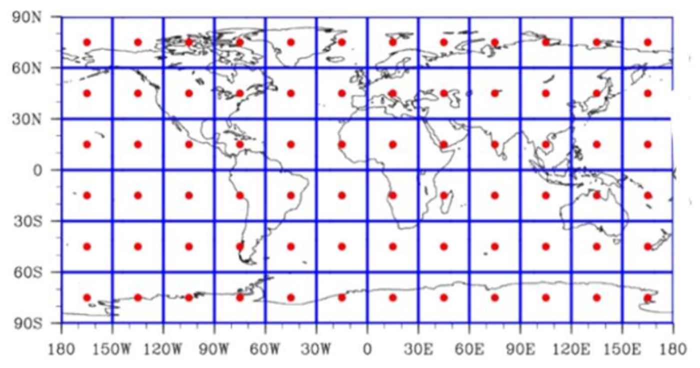


Figure1 Rectangular Equiangular Grid [6]

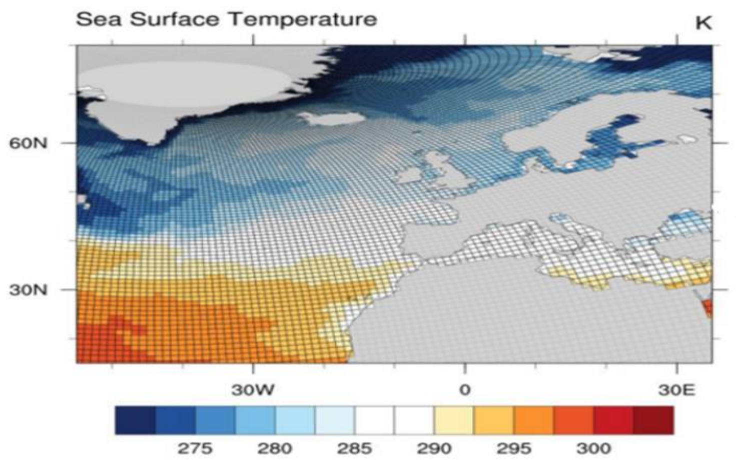


Figure 2 Curvilinear Grid MPI-ESM-LR (2D lat/lon arrays) [6]

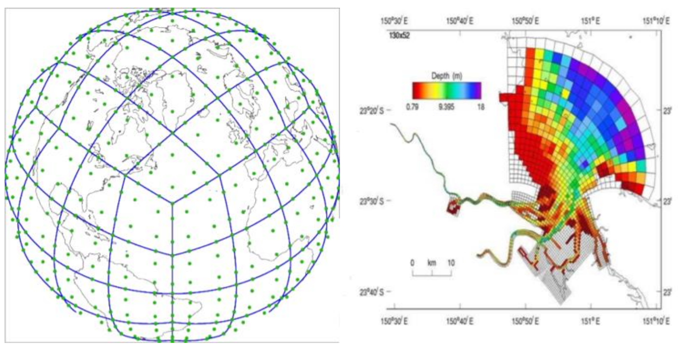


Figure3 Unstructured Cubed-Sphere and Unstructured Drainage/river [6]

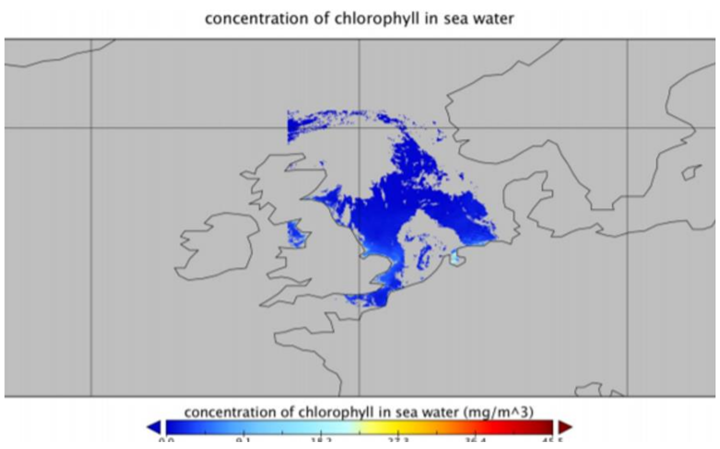

concentration of chlorophyll in sea water
concentration of chlorophyll in sea water (mg/m^3)

Figure 4 Swash Curvelinear [6]

Besides, as the model's accuracy increases, its data output becomes larger. For instance, the size of a 50-year dataset file is usually larger than five gigabytes. Data of this size are of huge challenge to memory space and can bring forth unacceptable running times.

Summarily, the current CMIP model outputs tend to have a large size and have more diverse formats. Meanwhile, the existence of CEDA and JASMIN can help in dealing with these model outputs. In this case, EZclim1.0, which can only run locally and cannot quickly process large data, becomes no longer applicable. Thus, EZclim2.0 is derived in this project to enhance the software's scope of application.

1.2 Existing Software (EZclim1.0)

EZclim1.0 was developed by Adanna Akwataghibe from ACSE-2018 to efficiently analyse the model outputs. It aims to shorten the gap between climate scientists and users with limited knowledge in climate science. In EZclim, the steps visible to users are: *Input data -> Select options -> Get output*.

EZclim1.0 was developed using Python, and combines the steps involving reading, pre-processing and calculating climate modelling diagnostics into one programme. Climate modelling diagnostics such as mean, median, standard deviation and root-mean-squared error can be calculated using EZclim1.0. All of these outputs are saved in NetCDF files, which can be visually represented using maps, histograms, time series and box plots. Moreover, the software can re-grid, extract regions, calculate climate indices and rotate the grid, amongst other functions.

EZclim1.0 also has many flaws as well. As mentioned, models in CMIP6 often use Swath or unstructured grids, whilst EZclim1.0 only accepts data having specific types. Furthermore, when using EZclim1.0, it takes a long time to download the datasets to a local repository. Likewise, if the dataset size is larger than 1 GB, it often takes much time to run the code, potentially running out of memory in the process.

Thus, EZclim2.0 is developed in this project to solve the aforementioned problems and to ultimately enhance the software's scope of application.

1.3 Summary and Objectives

The main objectives of this project are:

1. to port the software to JASMIN service;
2. to pre-process the dataset for acceptance by EZclim; and,
3. to parallel the EZclim code to attain a better performance.

Software Description

1. Installation and Dependency

Like EZclim1.0, EZclim2.0 is a stand-alone package, but is designed for use on JASMIN. A user should apply for a JASMIN account and utilise an SSH server to login to the JASMIN service [8]. To pass the account review, the user needs to provide relevant identification and his research projects in related fields. Afterward, he can download the EZclim2.0 package from GitHub and upload it in JASMIN. The EZclim2.0 should run successfully by then. The details on how to use EZclim2.0 can be reviewed on the user guide in the GitHub repository. Running on a local machine is possible, but needs further configuration; the steps on how to do so is also mentioned in the user guide.

2. Working Flow and Functionality

The overall working flow can be divided into three main parts, as illustrated in Figure 5.

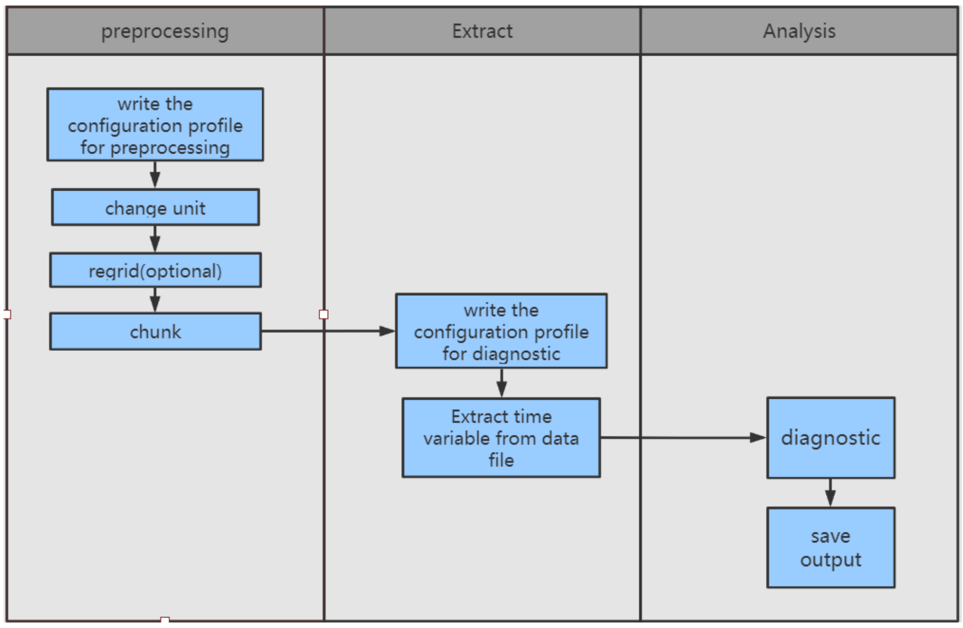

preprocessing
Extract
Analysis
write the configuration profile for preprocessing
change unit
regrid(optional)
chunk
write the configuration profile for diagnostic
Extract time variable from data file
diagnostic
save output

Figure 5 Working Flow Chart

Pre-processing is the first step. It initially processes data files for acceptance by the succeeding steps. A continuous software assessment yields that the root cause of data incompatibility is the data file’s grid, size and unit. The unit setback entails some variable units in the CMIP dataset not being identified by EZclim1.0. EZclim2.0 permits an input of a modified yet identifiable unit as a replacement to the unidentified one. The grid setback in EZclim1.0 is that CMIP6 grids are usually non-rectangular ones, which are not manageable. EZclim2.0 grants an interface solution for providing a rectangular model grid by re-gridding the archetype grid into the model one. The parallel technique openMP is likewise employed to accelerate processing in this step. Re-gridding is discretionary; it may be turned off if the grid is already a rectangular grid.

To prevent excessive parameters in the remaining functions, a specific control parameter to enable or disable the functions is not provided. The target unit can be fixed as the dataset's unit, and the chunked pieces can be fixed as 1 to simulate a “turn off” or disable effect. Chunking addresses size problems due to memory overuse as a consequence of huge datasets. The number and respective names of output pieces after chunking can be user-defined. Such functions can be substantially tailored. Significant parameters such as target, grid and number of pieces can be set beforehand prior to pre-processing. It can be said that pre-processing usually begins with configuring the parameters. The said functions are then directed to alter the unit, re-grid and chunk the dataset.

Chunking lies at the conclusion of pre-processing because many small pieces should be obtained by this stage. Placing the chunking step as the first one requires a separate unit change and re-gridding on each file, which is more difficult and time-consuming. The Implementation section provides the specifics on the execution of the above steps. EZclim2.0 obtains pertinent data from files as iris cube data through the user-defined options during the extract part. In the following analysis part, obtained data are computed and evaluated. Analyses outputs are then saved as NetCDF files. Plots can also be stored according to the user’s discretion. Both parts are also present in EZclim1.0.

3. Software Structure

Each functionality component is supported by several Python files. The software structure is demonstrated in Figure 6.

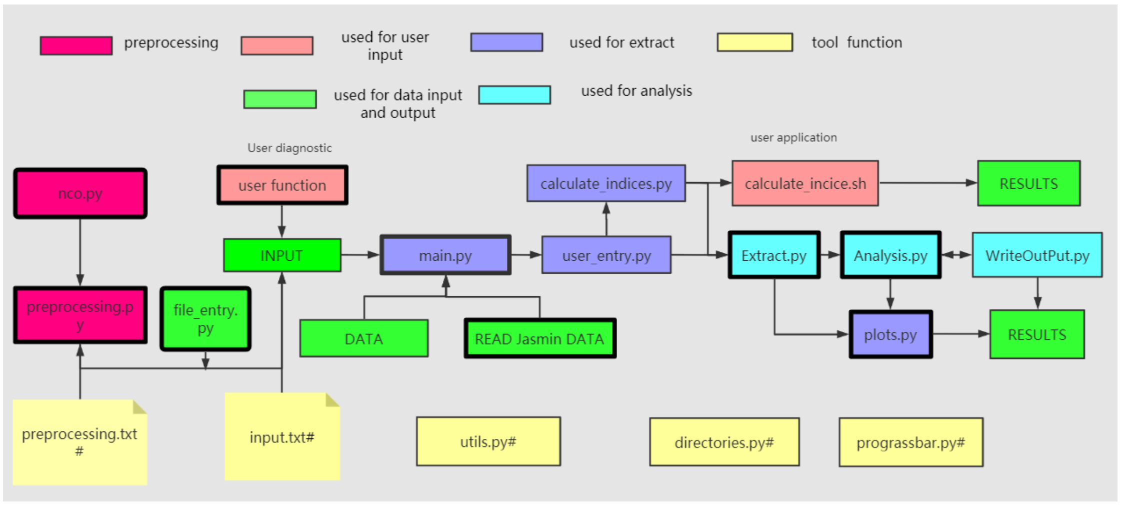


Figure 6 Structure of the Software

Files that surrounded with bold frame are either newly added files or modified ones in EZclim2.0. The *preprocessing.txt* file is used to configure the pre-processing option, whilst the *input.txt* file is utilised for the main function. The Python files in yellow are employed as tools in many other Python files. Taking *progressbar.py* as an example, it is used both in the pre-processing part and in the diagnostic part to create a progress bar. Each of the other remaining files, which are indicated with a small tag, corresponds to a functionality component. As mentioned, the aims of this project are to port to JASMIN, to pre-process and to perform parallelisation. The red part is responsible for pre-processing. In *nco.*py, the command-line style NCO operation is encapsulated into the Python code. The related functions are then used in *preprocessing.py*, which reads the pre-processing configuration from the *preprocessing.txt* file by calling *file_entry.py* and defining the working flow of the pre-processing part. The *progressbar.py* file is then added to EZclim2.0 to make up for the constraint that the user cannot use the built-in progress bar function of Python in JASMIN. Meanwhile, parallelisation is implemented on *Extract.py* and *Analysis.py*, for both parts are the most time-consuming in EZclim1.0. The details on how to integrate the codes to achieve the functionalities and on why these functionalities can be utilised to solve the three problems are presented in the next section.

Implementation and Code

1. Port to JASMIN

JASMIN is a shared resource for NERC's environmental science community and is needed to be applied in this project to use its resources [8]. The required datasets for EZclim2.0 are stored in Data Centre Archive [4]. As presented in the Figure 7 , JASMIN can only access data using the said virtual machine.

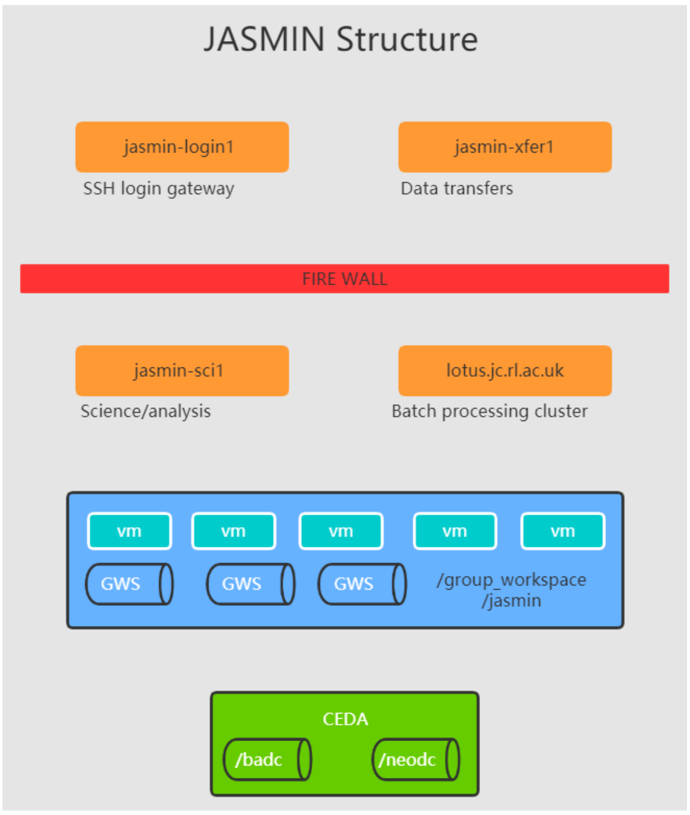


Figure 7 JASMIN Structure [8]

After the request to access is granted, the user can login to login-service in order to access the virtual machine. He can upload the whole EZclim1.0 package to JASMIN and use *bsub* to commit its task. The default Python version on JASMIN is Python2.7, but the user can load the *jaspy* module to enable Python 3. JASMIN does not support some third-party libraries that are used in EZclim1.0, like progress bar and seaborn. Considering that the pre-processing and diagnostic parts often take a long time to run, a self-creating progress bar is derived to indicate progress. Meanwhile, the plot functions of EZclim1.0 which are based on seaborn have been abandoned for two reasons. One is that the seaborn library is too complicated to be developed within three months; another is that the plot function may vary for different kinds of user functions. Hence, it is better if EZclim2.0 only provides the user with an interface to put its own user and plot functions. In EZclim2.0, the user can input its function into a specified file, when its function name should be configured as a user-function name in the configuration file. Afterward, he should set the user-function option in the configuration file to *True*. EZclim2.0 will then automatically run the user function with its corresponding plot function, which is similarly configured in the plot function file.

2. Pre-processing

For the pre-processing part, there are three main problems to be solved variable unit problem, grid problem, and memory problem. To solve these problems, the NetCDF Operators (NCO) is introduced. The NCO is a toolkit for manipulating and analysing NetCDF data. It is designed to be used on the command line, but this project implements a Python-style access to it. A complete NCO console command is chunked into pieces, whilst the user just needs to write the configuration file named *preprocess.txt*.

```
# REQUIRED ARGUMENTS

# ---------------------------------------------------------------------------------------------------------------------

File Path:

Number of pieces:

Number of year:

Number of time:

Variable Name:

Output Unit:

Model Grid:

Output Prefix:

Start_year:

output_folder:

regrid: False

# ----------------------------------------------------------------------------------------------------------------------
```

Figure 8 preprocess.txt

The parameters in yellow, green and blue are used for the three problems respectively, whilst the parameters in purple are used for the output. EZclim2.0 will read the required parameters from the configuration file and combine these parameters with the previous pieces in order to create a complete NCO console command. The NCO commands and their corresponding Python codes are presented below.

NCO command :

```
ncatted -O -a units,{variable},o,c,{output_unit} -o {local_file_path}  {filepath}
```

Python code:

```
string1 = 'units,'+self.variable+',o,c,'+self.output_unit
list = ['ncatted','-O','-a',string1,'-o',self.local_file_path]
command = list + [self.filepath]
```

Figure 9 Transfer NCO command to Python code

The blue part represents the parameters that are read from the configuration file. The details on how these parameters are combined with NCO to solve the three problems are discussed below.

Regarding the variable unit problem, some of the variable names stored in JASMIN cannot be recognised by EZclim1.0, because the outside library iris used in EZclim1.0 does not support some of the variable units or its variant. Taking 'salinity' as an example, some datasets use 0.001 as its unit, whilst others use PSU. The latter cannot be recognised by iris. In this case, EZclim2.0 forces the user to input a recognisable variable unit. If it is unknown whether a unit is recognisable or not, inputting numerical units like '0.001' is always a good choice. Afterward, the *ncatted* command of NCO is used to change the unit of the original file. Normally, *ncatted* changes the original file directly; however, since EZclim2.0 is ported to JASMIN, the data stored in JASMIN is not allowed to be changed. Thus, a changed copy of the original file becomes the output.

The second problem pertains to Swash, curvilinear and unstructured grids. EZclim1.0 only supports rectangular grids, whilst a lot of models in CMIP6 are using non-rectangular ones. The aim of the pre-processing part in this project is to remap the grids to normal rectangular ones in order to meet EZclim1.0's requirement. Below Figure 10 and 11 depict the effect of remapping. Figure 10 is the extreme examples to show the intuitive changes. If the resolution of re-grid increase, effect like what in Figure 11 will be obtained.

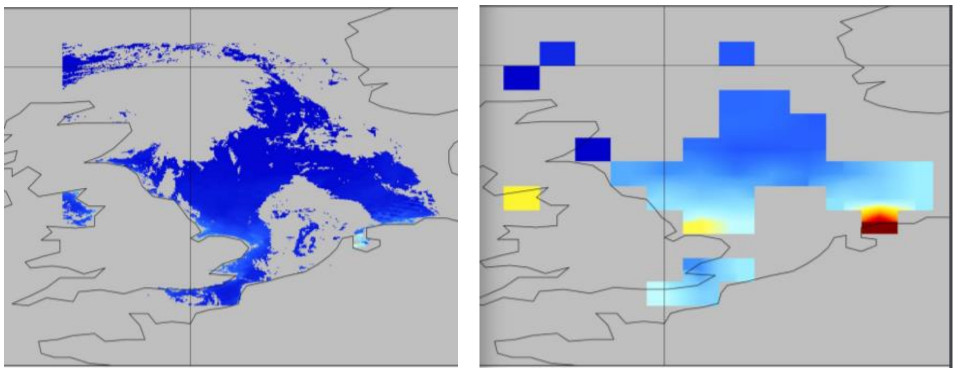


Figure 10 p1 Swash[6]   Figure 10 p2 Rectangular[6]

Figure 10 remap from swash to rectangular

The two pictures in Figure 11 look quite similar but note the red part in p2, the distortion is a straight line in geographical coordinates.

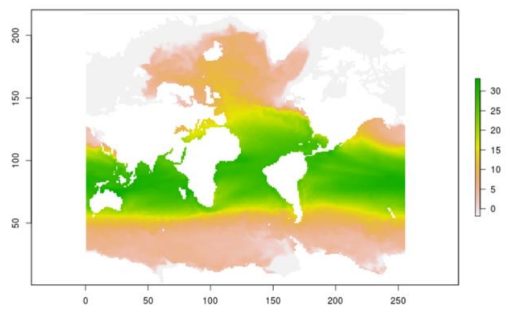


Figure 11 p1 Swash[19]

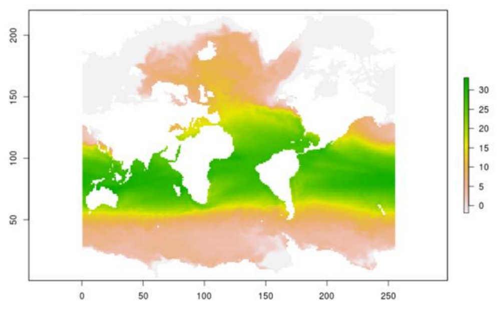


Figure 11 p2 Rectangular[19]

Figure 11 remap from swash to rectangular

The function first finds the provided model grid files or map files from the model grid path, which are provided in the configuration file. The grid details that are provided in the CF metadata of the data file itself are then supplemented to this information. Then, the function tests the known rectangular grid types – equidistant, FV, offset, Gaussian – and supplements the metadata with the exact information, which are derived from the inferred grid type [6]. The core of this algorithm is data interpolation transformation of the different grid projections. However, *ncremap* must deduce the central position of the grid attribute unit from them. Thus, the said function can only remap complicated grids, like Swath grid to rectangular grid, but not the other way around.

The third problem is the memory problem. The available memory in JASMIN is usually limited for general users. Memory not only refers to actual storage memory, but also includes running memory. When dealing with CMIP6 data, because of its size, the available running memory is always insufficient. EZclim2.0 solves this problem by chunking the original data file into smaller pieces. These data pieces will each assign an EZclim2.0 task and be submitted to the queue separately, whilst their outputs can be recombined into a complete result. The chunking function uses time as a basis for segmentation and uses the *ncks* operation of NCO to cut the file. The output format of the file can be defined by the purple part of the configuration file. One reason why the number of time and years should be the input is that various CMIP models use different time scales to represent time.

3. Parallelisation

The parallelisation part is divided into two parts. They are the parallelization of the original EZclim1.0 part and parallelization of the pre-processing process.

To achieve the parallelisation of the EZclim1.0, a multiprocessing library is introduced. A multiprocessing package has a similar api with a threading module and supports spawning processes. The difference is that the multiprocessing module offers a convenient means for parallelising the execution of a function across multiple input values, hence distributing the input data across processes known as data parallelism [20]. A typical multiprocessing parallel code is demonstrated in Figure 12.

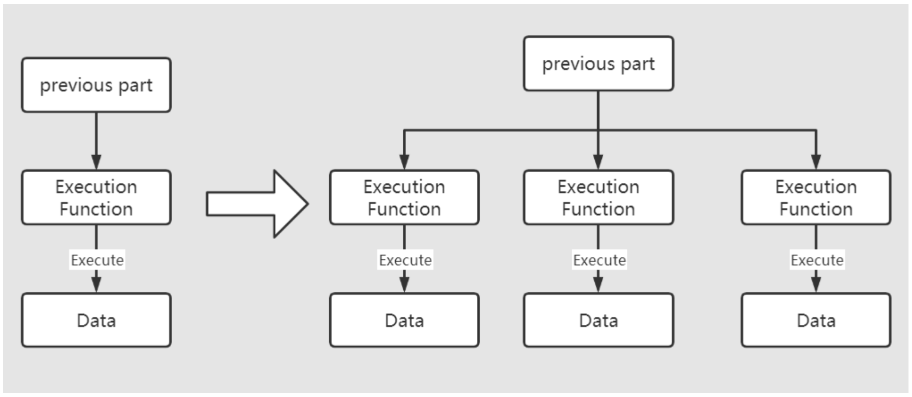


Figure 12 Typical Multiprocessing Parallel Mode

Considering this, the next step is to separate the operation and encapsulate it into a function. Indeed, the most time-consuming parts in EZclim2.0 are the data extracting and the analysis parts. Thus, both parts are encapsulated into two independent functions. Afterward, the data are chunked based on the threading number provided and are manipulated separately by the functions. Then, the most suitable number of threads are tested. The Figure13 presents the relationship between the running time and the number of threads.

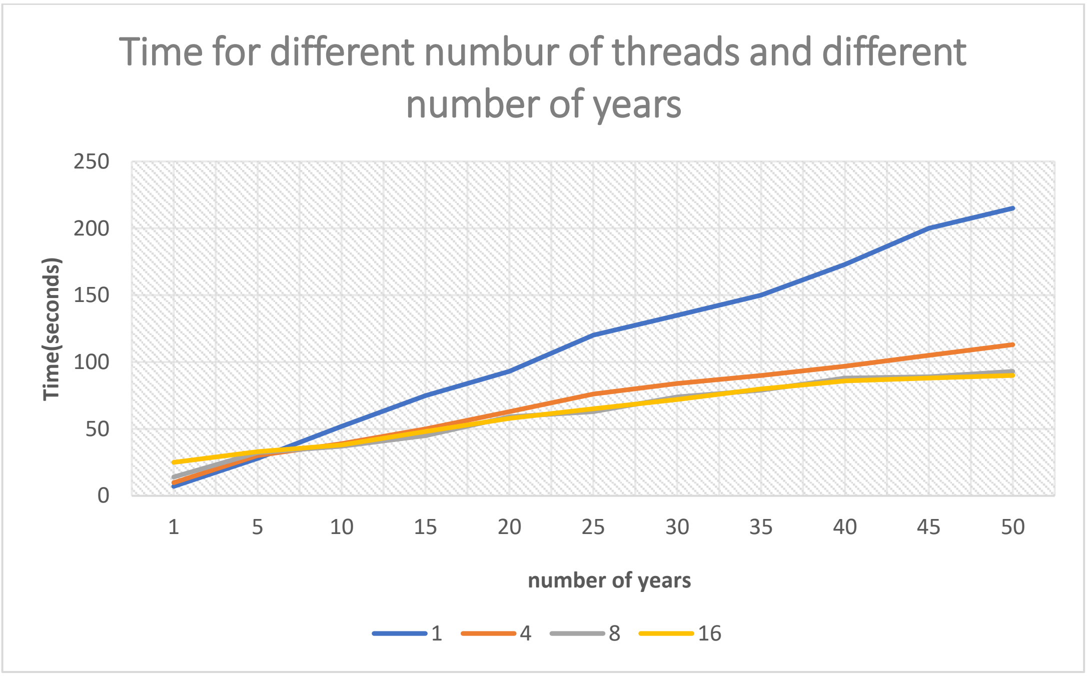


Figure 13 Time Cost for different number of threads and different number of years(different colour represent different number of threads)

The figure above illustrates that parallelization performance for a small (1-5 years) dataset is even worse at start. It then depicts a direct linear relationship between time cost of serial mode and number of years. Mode 4 is similar to the serial mode, but with less gradient. Modes 8 and 16 likewise have an initially increasing tendency, with gradually flat curves shortly afterward. This is because thread generation and switching imply time cost. In counterbalancing starting thread consumption, if parallelism is insufficient, serial mode can serve as a better option. As the size of data intensifies, parallelization is more advantageous due to negligible time cost for starting and grid switching. Among the modes, 8-thread is selected. It does not seem too extended for managing small datasets, yet still exhibits decent execution while processing big ones.

Pre-processing parallelization equates to re-gridding parallelization. Small datasets tend to have a rapid re-gridding method. For large ones like CMIP6, re-gridding typically takes around 10 to 20 minutes. This duration is quite long despite the re-gridding process occurring only once. NCO renders several choices for ncremap in this case.

```
ncremap … -p nil …     # Serial
ncremap … -p bck ...   # Background (default)
ncremap … -p mpi …   # MPI
```

Figure 14 different format ncremap Command

Serial, background and MPI are the three operating modes supported by ncremap. Background mode is the default, especially if -p is unidentified. The operation divides the steps for every remap into one node for the background mode.

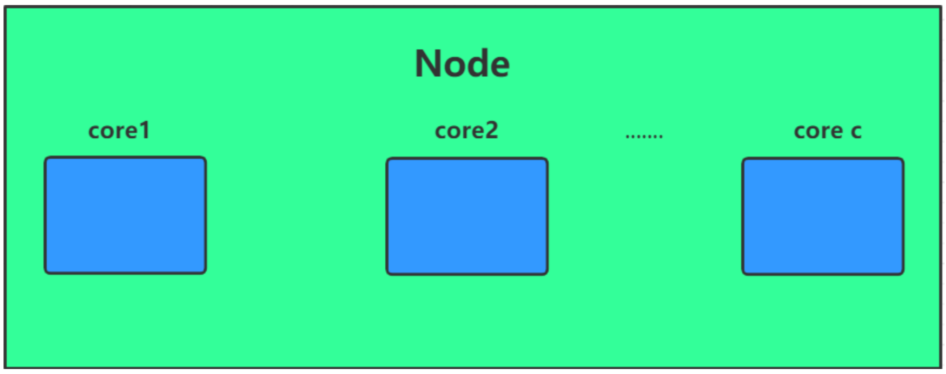


Figure 15 Background Mode

Unlike background mode, the MPI operation will spawn MPI process to new node for each remap. MPI serves as a communication protocol for computers to do parallel programming. It assists in communication, whether point-to-point or collective. NCO aids in parallel computing in this mode using MPI to do parallel computing by calling ncremap ….-p mpi.

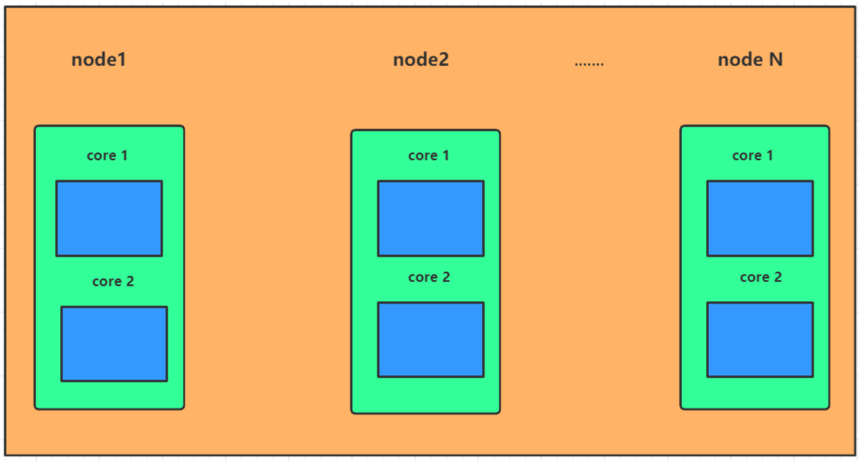


Figure 15 MPI Mode

Runtime is immensely mitigated when MPI parallelization is applied, but one drawback is the increase in RAM usage. Such disadvantage can cause several mid-sized activities not to run. MPI parameters and RAM demand have a relationship as depicted in the following formula:

RAM = sizeof(map) + ~4x sizeof(var) per thread

It is discussed that inadequate RAM is definitely a huge predicament when considering large datasets. This becomes graver when MPI mode is applied. This mode is applicable for small or moderate structures that require fast high-resolution remapping. Serial mode serves as a substitute especially for laptops and desktops that have small RAM, but further slows down the software. Both are improper in this project.

Background mode can perform more suitably in this case because it operates on node computations, login nodes and big datasets. Along with threading parallelization, it is selected, and overall deemed as openMP. It operates on only one node unlike MPI and employs shared memory. It can still encounter RAM demands, albeit rare. There are two approaches for resolving this. The first is by chunking prior to ncremap, then every piece is applied with ncremap. This takes less time yet employs tedious implementation. The second is to determine the equilibrium between threads and RAM. Various trials were conducted on the so_Omon_HadCM3_decadal1982_r1i2p1_198211-199212.nc dataset.

Table 1 Parallel	NCO(serial)	NCO(threads=8)	NCO(threads =16)
1 year	0m20s-0m35s	0m06s -0m07s	0m05s – 0m06s
5 year	1m30s – 1m40s	0m12s- 0m13s	0m10s -0m11s
25 year	5m00s – 7m30s	0m48s – 0m50s	0m45s – 0m50s
50 year	14m50s -24m50s	2m00s-2m05s	fail

Table 1

Re-grid time on different size of so_Omon_HadGEM3-GC31-LL_historical_r1i1p1f3_gn_185001-189912.nc

Table 1 shows mode 8 attained rational capability and did not encounter RAM difficulties. Yet another test negated this finding. It thus seems the 16-thread option is the most viable. It is inferred that number of threads is linked with dataset size, but additional tests are not conducted due to time constraints. Greater time allotment can have this project determine the linkage between parameters and dataset size. Considering this project, the time constraint only occurred from the CMIP6 dataset, which is similar to the case of Table 1, Mode 8 is eventually selected.

4. Testing

Function tests are applied in all discussed operations. Various dataset variables and grids are utilized in this case. Actual users are also made to do the testing. An initial critique from them involves a lengthy re-gridding and no indication of a working or failed program.

Further modifications are then applied. A progress bar function was generated to determine project progress and improve user experience. This is to enable users to determine the program’s workability and projected completion duration. To speed up re-gridding, NCO enables utilisation of parallelization algorithms. The existing version with grid parallel algorithm was eventually created after further tests.

Discussion

This section provides a short project discussion. It particularly points out researchers' views, difficulties faced, preventable faults, and potential enhancements.

1. Why is NCO selected for data file operation?

   This is primarily performed to manage netCDF data. There are several available third-party netCDF packages aside from NCO. They include Climate Data Operators (CDO) and NCAR Command Language (NCL) [18]. NCL is an interpreted language; others provide a command-line operator compilation [11][17]. All three have similar functions despite contrasting formats [9][10][11][18]. It all boils down to performance as the most vital. Re-gridding is the most laborious during pre-processing stage. Many performance trials are executed in this case.

Dataset	NCL	CDO	NCO
so_Omon_HadGEM3-GC31-LL_historical_r1i1p1f3_gn_185001-189912.nc(5.1GB)	20 min 10 s - 50 min 09s	13 min 33s - 15 min 20 s	7 min 30 s - 10 min 40s
so_Omon_HadCM3_decadal1982_r1i2p1_198211-199212.nc(404.8 MB)	1min 55 s - 2 min 00s	0 min 10 - 0 min 13	0 min 10s - 0 min 11 s

Table 2 Performance Test on NCL CDO NCO

   Above Table 2 indicate that, regardless of dataset size, NCO outperforms the two other packages. This package is eventually selected.

2. Migrating to EZclim2.0 from EZclim1.0

   The necessity and gains from EZclim2.0 are already established earlier. These are counteracted by the cost in migrating from Ezclim1.0. Both sides are usually of huge concern through the project's progress.

   Only some external Python libraries are supported by JASMIN. This does not include EZclim1.0-compatible ones such as progress bar and seaborn. It requires a choice if these functions should be forbidden or if they can be created separately instead. This is exemplified by the plot function, which is replaced by an independent progress bar function during implementation.

   Another predicament has occurred when NCO's ncremap function is not compatible to Windows. This issue leads to irregular CMIP6 datasets remain unsettled. As the situation can perform only on Linux, convenience is somehow lost, but still acceptable. This can be typically addressed through managing and evaluation climate outputs using a data platform. The user may also pre-process through JASMIN, obtain the outputs, then execute the remaining codes locally.

3. Potential enhancement

   The outcomes of analyses can be innately provided using a plot. This is not realized in this project because seaborn is not supported in JASMIN. The said predicament may be addressed by granting an interface for inputting modifications, including plot function.

A related challenge is that several climate modelers use seaborn as well for plotting. The said package has many beneficial plotting functions, despite itself not designed for the climate model output. This can be suitably addressed by constructing valuable plotting tools particularly for climate model output. One applicable plotting tool is a box plot. It is typically employed for analysing climate outputs and finding daily variations in air temperature in a given month. This function can save users much time to derive their own plot function.

Conclusion

EZclim1.0 was assessed in many aspects during this project. Various resolutions were provided to address its limitations. This has paved the way for creating EZclim2.0 to improve user experience by working on reasonable suggestions from different feedback. It has eventually attained three goals. It can be ported to JASMIN, it can operate pre-processing before diagnostics, and it can apply parallelization algorithms. This enhanced software version can manage various dataset types and process data faster. Modifications are also possible for pre-processing steps. A progress bar is also provided to display the software's current status. EZclim2.0 represents a big step forward in terms of functionalities in the context of CMIP experiments. Future improvement could be designing a plot tool function to help user plot their analysis result.

Bibliography

[1] Australian Academy of Science. What is Climate change? https://www.science.org.au/learning/general-audience/science-climate-change/1-what-is-climate-change, 2019.

[2] Carbon Brief. How do climate models work? https://www.carbonbrief.org/qa-how-do-climate-models-work, 2018

[3] Steven J. Phipps. Coupled climate system modelling. https://www.stevenphipps.com/teaching/coe2013.pdf 2013

[4] CEDA. https://www.ceda.ac.uk/about/what-we-do/

[5] Dr. Charles S. Zender , Dr. Christopher S. Lynnes , and Walter Baskin. Easy Access to and Analysis of NASA and Model Swath-like Data . http://dust.ess.uci.edu/prp/prp_axs/prp_axs.pdf

[6] Charlie Zender. How to Regrid Swash Data? http://dust.ess.uci.edu/smn/smn_nco_gsfc_201604.pdf, 2016

[7] How to find data https://docs.esmvaltool.org/projects/ESMValCore/en/latest/quickstart/find_data.html

[8] JASMIN https://accounts.jasmin.ac.uk/

[9] NCO http://nco.sourceforge.net/

[10] CDO https://code.mpimet.mpg.de/projects/cdo/

[11] NCL https://www.ncl.ucar.edu/Document/

[12] UK climate projections. https://www.metoffice.gov.uk/research/collaboration/ukcp, 2019.

[13] How do climate model works. https://www.carbonbrief.org/qa-how-do-climate-models-work

[14] CDO vs NCO. http://hannahlab.org/cdo-vs-nco/

[15] Schulzweida, Uwe. CDO User Guide. https://doi.org/10.5281/zenodo.2558193, February 2019

GitHub repository: https://github.com/acse-2019/irp-acse-yz219

[16] UK climate projections. https://www.metoffice.gov.uk/research/collaboration/ukcp, 2019.

[17] Third-party netCDF packages: NCO, NCL, CDO https://www.unidata.ucar.edu/software/netcdf/workshops/most-recent/third_party/index.html

[18] Easy Access to and Analysis of NASA and Model Swath-Like Data https://earthdata.nasa.gov/esds/competitive-programs/access/model-swath-like-data

[19] Mark R Payne, DTU-Aqua, Charlottenlund, Denmark https://rpubs.com/markpayne/132500

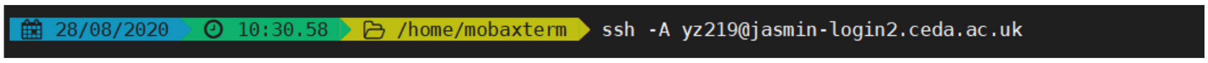
[20] what is parallelization ? https://www.computerhope.com/jargon/p/parallelization.htm#:~:text=Parallelization%20is%20the%20act%20of,the%20next%2C%20then%20the%20next.

Appendix

Demonstration of the EZcim2.0 Running Process

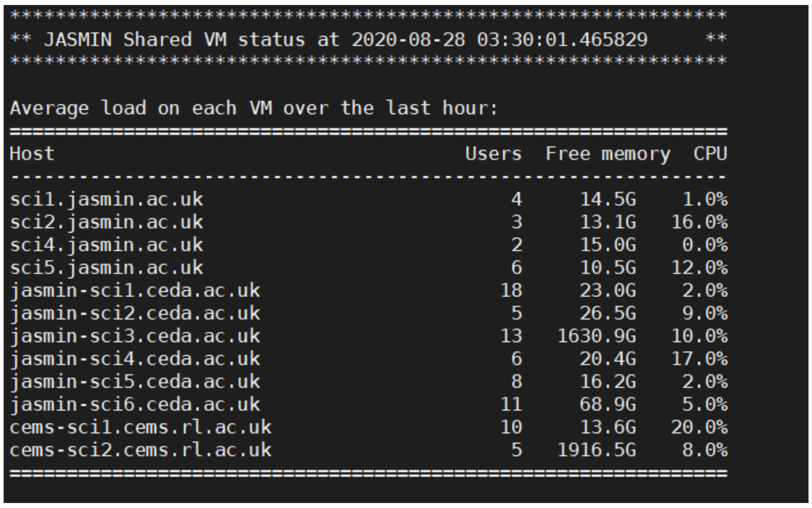
1.1 Login to JASMIN

SSH to the JASMIN login server



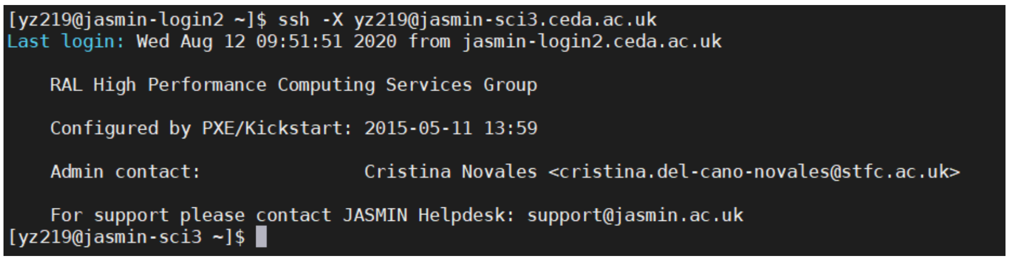
```
28/08/2020  10:30.58  /home/mobaxterm  ssh -A yz219@jasmin-login2.ceda.ac.uk
```

GET the list of available VMs



```
***************************************************************
** JASMIN Shared VM status at 2020-08-28 03:30:01.465829     **
***************************************************************

Average load on each VM over the last hour:
===============================================================
Host                                    Users  Free memory  CPU
---------------------------------------------------------------
sci1.jasmin.ac.uk                           4      14.5G    1.0%
sci2.jasmin.ac.uk                           3      13.1G   16.0%
sci4.jasmin.ac.uk                           2      15.0G    0.0%
sci5.jasmin.ac.uk                           6      10.5G   12.0%
jasmin-sci1.ceda.ac.uk                     18      23.0G    2.0%
jasmin-sci2.ceda.ac.uk                      5      26.5G    9.0%
jasmin-sci3.ceda.ac.uk                     13    1630.9G   10.0%
jasmin-sci4.ceda.ac.uk                      6      20.4G   17.0%
jasmin-sci5.ceda.ac.uk                      8      16.2G    2.0%
jasmin-sci6.ceda.ac.uk                     11      68.9G    5.0%
cems-sci1.cems.rl.ac.uk                    10      13.6G   20.0%
cems-sci2.cems.rl.ac.uk                     5    1916.5G    8.0%
===============================================================
```

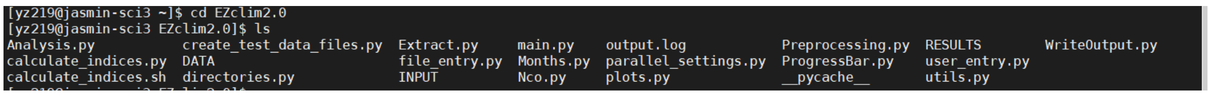
Choose one of the VM



```
[yz219@jasmin-login2 ~]$ ssh -X yz219@jasmin-sci3.ceda.ac.uk
Last login: Wed Aug 12 09:51:51 2020 from jasmin-login2.ceda.ac.uk

    RAL High Performance Computing Services Group

    Configured by PXE/Kickstart: 2015-05-11 13:59

    Admin contact:              Cristina Novales <cristina.del-cano-novales@stfc.ac.uk>

    For support please contact JASMIN Helpdesk: support@jasmin.ac.uk
[yz219@jasmin-sci3 ~]$
```

CD EZclim2.0 which have been uploaded to JASMIN

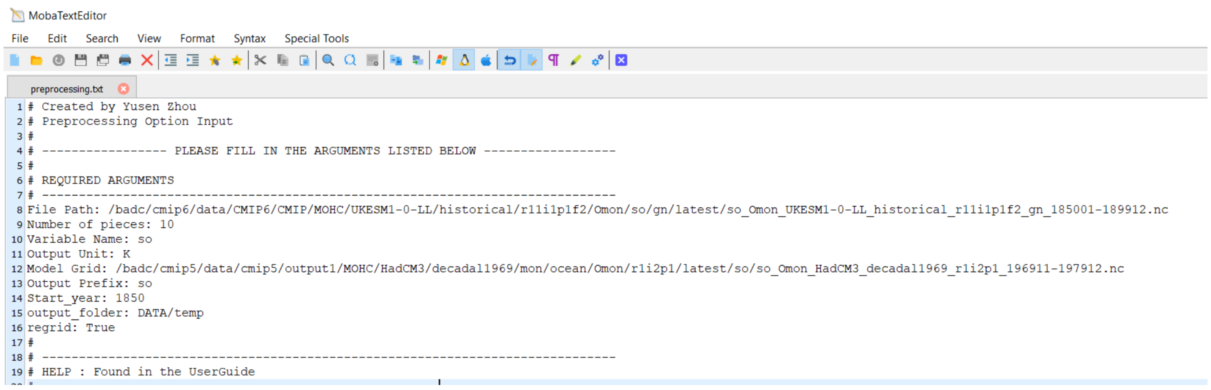
GitHub repository: https://github.com/acse-2019/irp-acse-yz219



```
[yz219@jasmin-sci3 ~]$ cd EZclim2.0
[yz219@jasmin-sci3 EZclim2.0]$ ls
Analysis.py           create_test_data_files.py  Extract.py     main.py    output.log            Preprocessing.py  RESULTS        WriteOutput.py
calculate_indices.py  DATA                       file_entry.py  Months.py  parallel_settings.py  ProgressBar.py    user_entry.py
calculate_indices.sh  directories.py             INPUT          Nco.py     plots.py              __pycache__       utils.py
```

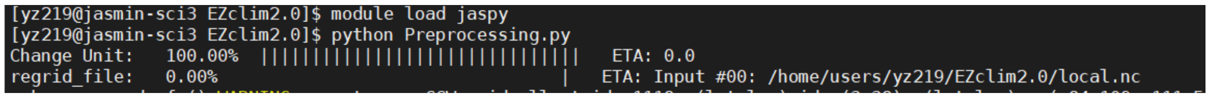
1.2 Do Pre-processing

Modify the preprocess.txt file

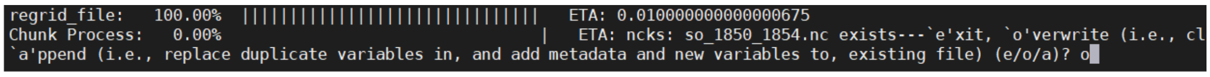


```
# Created by Yusen Zhou
# Preprocessing Option Input
#
# ---------------- PLEASE FILL IN THE ARGUMENTS LISTED BELOW -----------------
#
# REQUIRED ARGUMENTS
# ---------------------------------------------------------------------------
File Path: /badc/cmip6/data/CMIP6/CMIP/MOHC/UKESM1-0-LL/historical/r11i1p1f2/Omon/so/gn/latest/so_Omon_UKESM1-0-LL_historical_r11i1p1f2_gn_185001-189912.nc
Number of pieces: 10
Variable Name: so
Output Unit: K
Model Grid: /badc/cmip5/data/cmip5/output1/MOHC/HadCM3/decadal1969/mon/ocean/Omon/r1i2p1/latest/so/so_Omon_HadCM3_decadal1969_r1i2p1_196911-197912.nc
Output Prefix: so
Start_year: 1850
output_folder: DATA/temp
regrid: True
#
# ---------------------------------------------------------------------------
# HELP : Found in the UserGuide
```

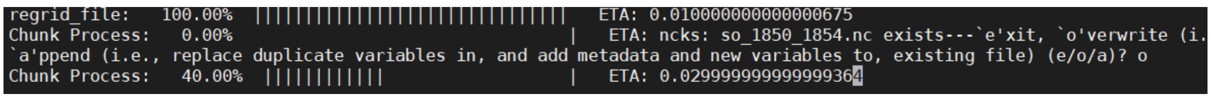
Run Preprocessing.py and wait the program to finish



```
[yz219@jasmin-sci3 EZclim2.0]$ module load jaspy
[yz219@jasmin-sci3 EZclim2.0]$ python Preprocessing.py
Change Unit:    100.00%  |||||||||||||||||||||||||||||   ETA: 0.0
regrid_file:    0.00%                                  |    ETA: Input #00: /home/users/yz219/EZclim2.0/local.nc
```



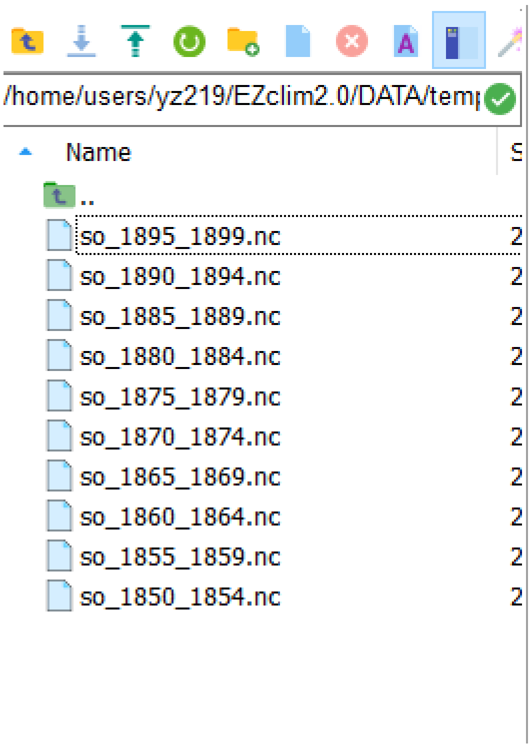
```
regrid_file:    100.00%  ||||||||||||||||||||||||||||   ETA: 0.010000000000000675
Chunk Process:    0.00%                                |    ETA: ncks: so_1850_1854.nc exists---`e'xit, `o'verwrite (i.e., cl
`a'ppend (i.e., replace duplicate variables in, and add metadata and new variables to, existing file) (e/o/a)? o
```



```
regrid_file:    100.00%  |||||||||||||||||||||||||||||   ETA: 0.010000000000000675
Chunk Process:    0.00%                                |    ETA: ncks: so_1850_1854.nc exists---`e'xit, `o'verwrite (i.
`a'ppend (i.e., replace duplicate variables in, and add metadata and new variables to, existing file) (e/o/a)? o
Chunk Process:    40.00%  |||||||||||                  |    ETA: 0.029999999999999364
```

After the program finish we get all the files we need at the Data directories.

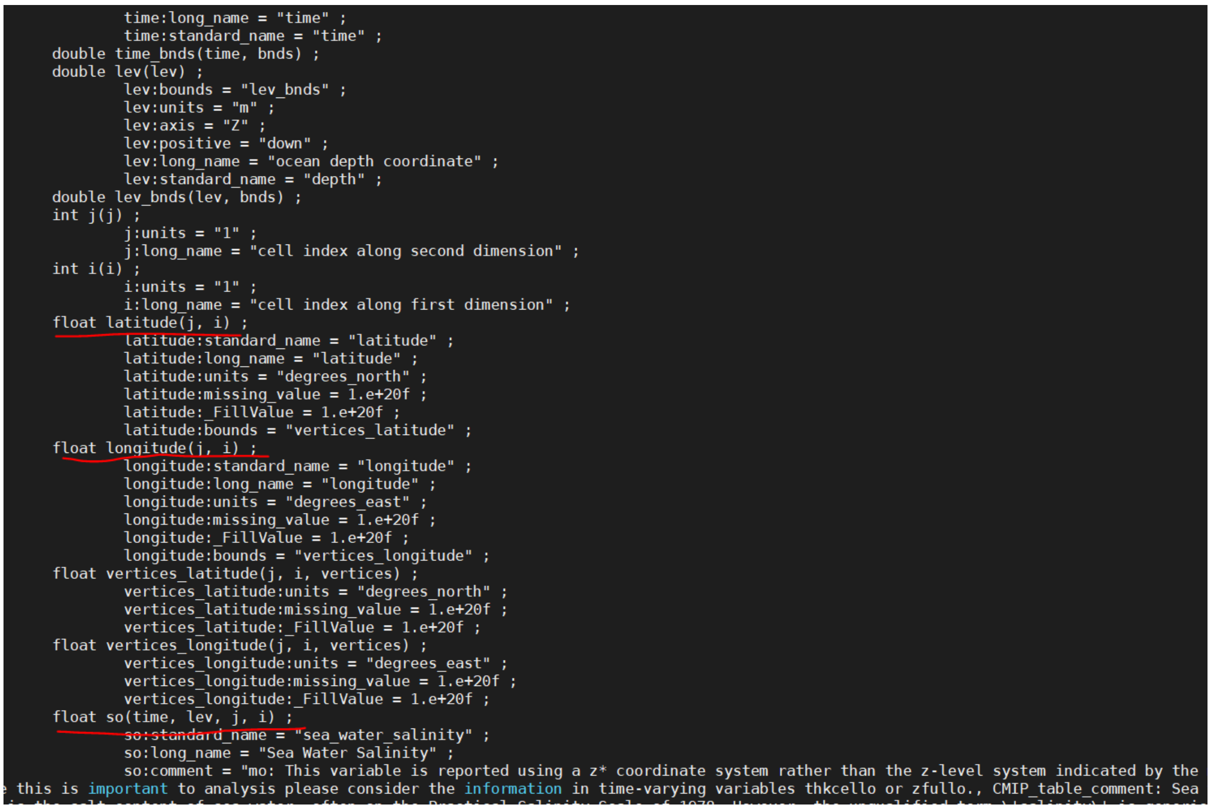
GitHub repository: https://github.com/acse-2019/irp-acse-yz219



```
/home/users/yz219/EZclim2.0/DATA/temp

Name                   S
..
so_1895_1899.nc        2
so_1890_1894.nc        2
so_1885_1889.nc        2
so_1880_1884.nc        2
so_1875_1879.nc        2
so_1870_1874.nc        2
so_1865_1869.nc        2
so_1860_1864.nc        2
so_1855_1859.nc        2
so_1850_1854.nc        2
```

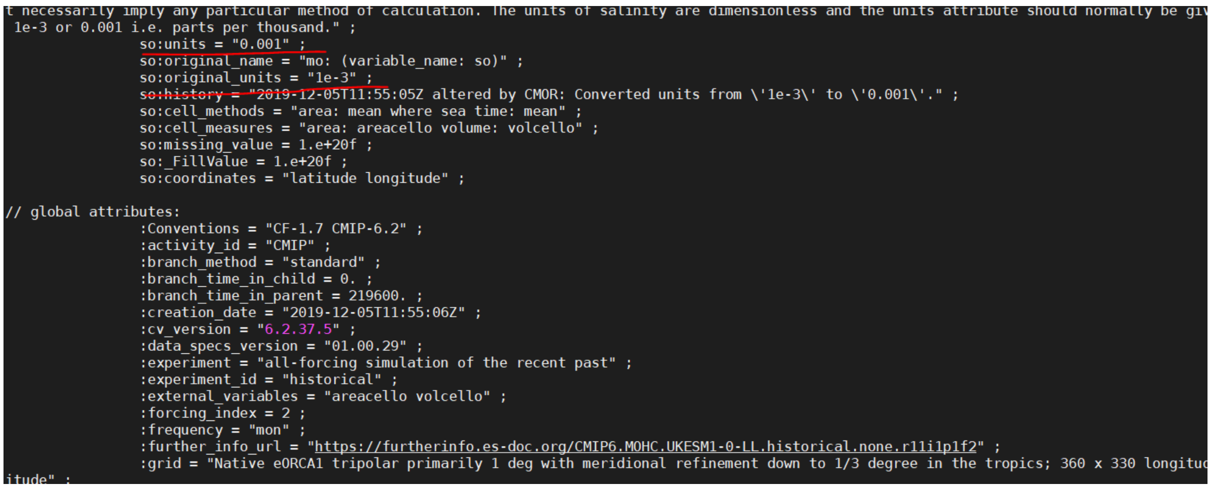
Use ncdump to compare the previous NetCDF file and current NetCDF file

Original



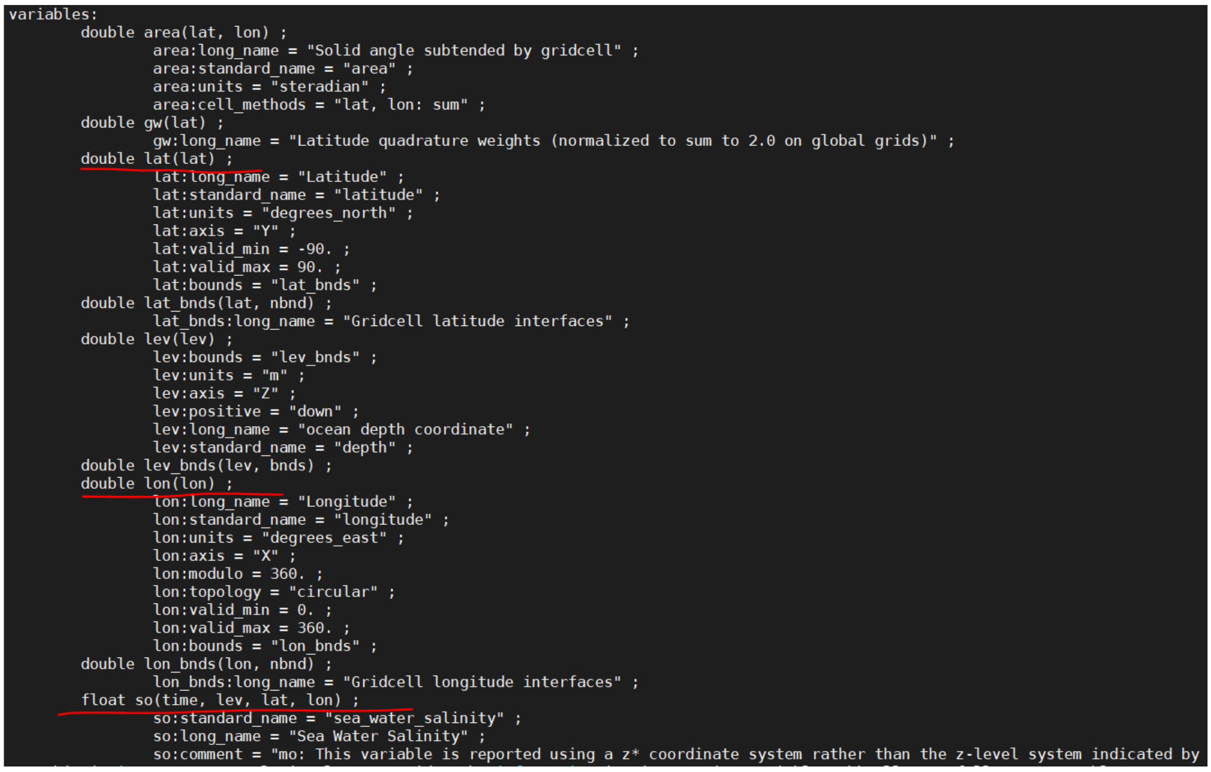
```
                time:long_name = "time" ;
                time:standard_name = "time" ;
        double time_bnds(time, bnds) ;
        double lev(lev) ;
                lev:bounds = "lev_bnds" ;
                lev:units = "m" ;
                lev:axis = "Z" ;
                lev:positive = "down" ;
                lev:long_name = "ocean depth coordinate" ;
                lev:standard_name = "depth" ;
        double lev_bnds(lev, bnds) ;
        int j(j) ;
                j:units = "1" ;
                j:long_name = "cell index along second dimension" ;
        int i(i) ;
                i:units = "1" ;
                i:long_name = "cell index along first dimension" ;
        float latitude(j, i) ;
                latitude:standard_name = "latitude" ;
                latitude:long_name = "latitude" ;
                latitude:units = "degrees_north" ;
                latitude:missing_value = 1.e+20f ;
                latitude:_FillValue = 1.e+20f ;
                latitude:bounds = "vertices_latitude" ;
        float longitude(j, i) ;
                longitude:standard_name = "longitude" ;
                longitude:long_name = "longitude" ;
                longitude:units = "degrees_east" ;
                longitude:missing_value = 1.e+20f ;
                longitude:_FillValue = 1.e+20f ;
                longitude:bounds = "vertices_longitude" ;
        float vertices_latitude(j, i, vertices) ;
                vertices_latitude:units = "degrees_north" ;
                vertices_latitude:missing_value = 1.e+20f ;
                vertices_latitude:_FillValue = 1.e+20f ;
        float vertices_longitude(j, i, vertices) ;
                vertices_longitude:units = "degrees_east" ;
                vertices_longitude:missing_value = 1.e+20f ;
                vertices_longitude:_FillValue = 1.e+20f ;
        float so(time, lev, j, i) ;
                so:standard_name = "sea_water_salinity" ;
                so:long_name = "Sea Water Salinity" ;
                so:comment = "mo: This variable is reported using a z* coordinate system rather than the z-level system indicated by the
e this is important to analysis please consider the information in time-varying variables thkcello or zfullo., CMIP_table_comment: Sea
```

GitHub repository: https://github.com/acse-2019/irp-acse-yz219



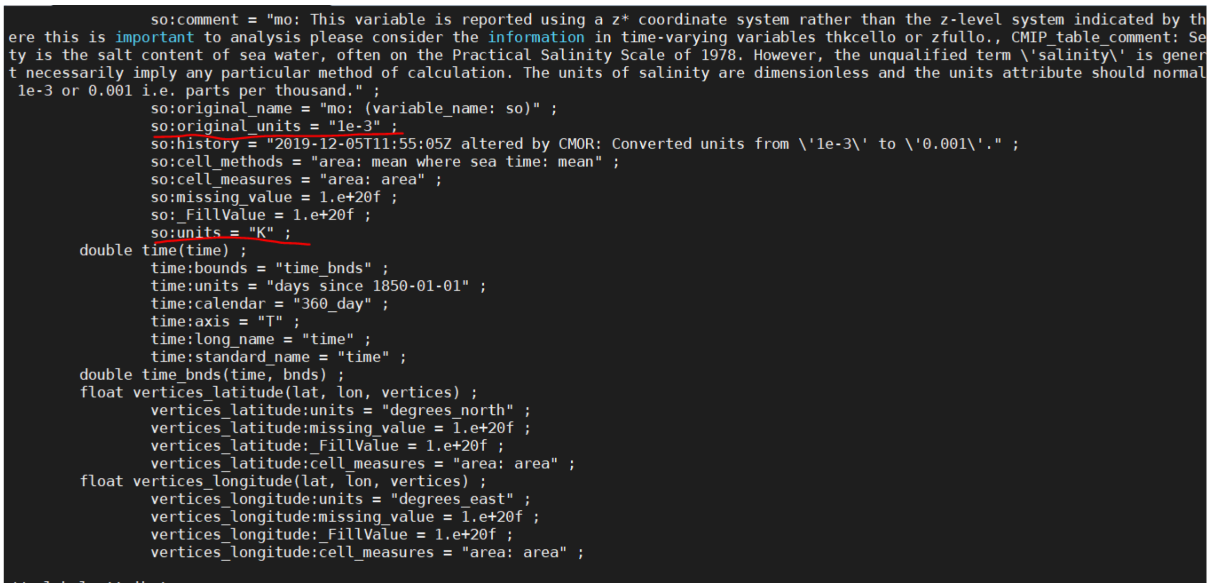
```
t necessarily imply any particular method of calculation. The units of salinity are dimensionless and the units attribute should normally be giv
 1e-3 or 0.001 i.e. parts per thousand." ;
                so:units = "0.001" ;
                so:original_name = "mo: (variable_name: so)" ;
                so:original_units = "1e-3" ;
                so:history = "2019-12-05T11:55:05Z altered by CMOR: Converted units from \'1e-3\' to \'0.001\'." ;
                so:cell_methods = "area: mean where sea time: mean" ;
                so:cell_measures = "area: areacello volume: volcello" ;
                so:missing_value = 1.e+20f ;
                so:_FillValue = 1.e+20f ;
                so:coordinates = "latitude longitude" ;

// global attributes:
                :Conventions = "CF-1.7 CMIP-6.2" ;
                :activity_id = "CMIP" ;
                :branch_method = "standard" ;
                :branch_time_in_child = 0. ;
                :branch_time_in_parent = 219600. ;
                :creation_date = "2019-12-05T11:55:06Z" ;
                :cv_version = "6.2.37.5" ;
                :data_specs_version = "01.00.29" ;
                :experiment = "all-forcing simulation of the recent past" ;
                :experiment_id = "historical" ;
                :external_variables = "areacello volcello" ;
                :forcing_index = 2 ;
                :frequency = "mon" ;
                :further_info_url = "https://furtherinfo.es-doc.org/CMIP6.MOHC.UKESM1-0-LL.historical.none.r11i1p1f2" ;
                :grid = "Native eORCA1 tripolar primarily 1 deg with meridional refinement down to 1/3 degree in the tropics; 360 x 330 longitud
itude" ;
```

After



```
variables:
        double area(lat, lon) ;
                area:long_name = "Solid angle subtended by gridcell" ;
                area:standard_name = "area" ;
                area:units = "steradian" ;
                area:cell_methods = "lat, lon: sum" ;
        double gw(lat) ;
                gw:long_name = "Latitude quadrature weights (normalized to sum to 2.0 on global grids)" ;
        double lat(lat) ;
                lat:long_name = "Latitude" ;
                lat:standard_name = "latitude" ;
                lat:units = "degrees_north" ;
                lat:axis = "Y" ;
                lat:valid_min = -90. ;
                lat:valid_max = 90. ;
                lat:bounds = "lat_bnds" ;
        double lat_bnds(lat, nbnd) ;
                lat_bnds:long_name = "Gridcell latitude interfaces" ;
        double lev(lev) ;
                lev:bounds = "lev_bnds" ;
                lev:units = "m" ;
                lev:axis = "Z" ;
                lev:positive = "down" ;
                lev:long_name = "ocean depth coordinate" ;
                lev:standard_name = "depth" ;
        double lev_bnds(lev, bnds) ;
        double lon(lon) ;
                lon:long_name = "Longitude" ;
                lon:standard_name = "longitude" ;
                lon:units = "degrees_east" ;
                lon:axis = "X" ;
                lon:modulo = 360. ;
                lon:topology = "circular" ;
                lon:valid_min = 0. ;
                lon:valid_max = 360. ;
                lon:bounds = "lon_bnds" ;
        double lon_bnds(lon, nbnd) ;
                lon_bnds:long_name = "Gridcell longitude interfaces" ;
        float so(time, lev, lat, lon) ;
                so:standard_name = "sea_water_salinity" ;
                so:long_name = "Sea Water Salinity" ;
                so:comment = "mo: This variable is reported using a z* coordinate system rather than the z-level system indicated by
```

GitHub repository: https://github.com/acse-2019/irp-acse-yz219



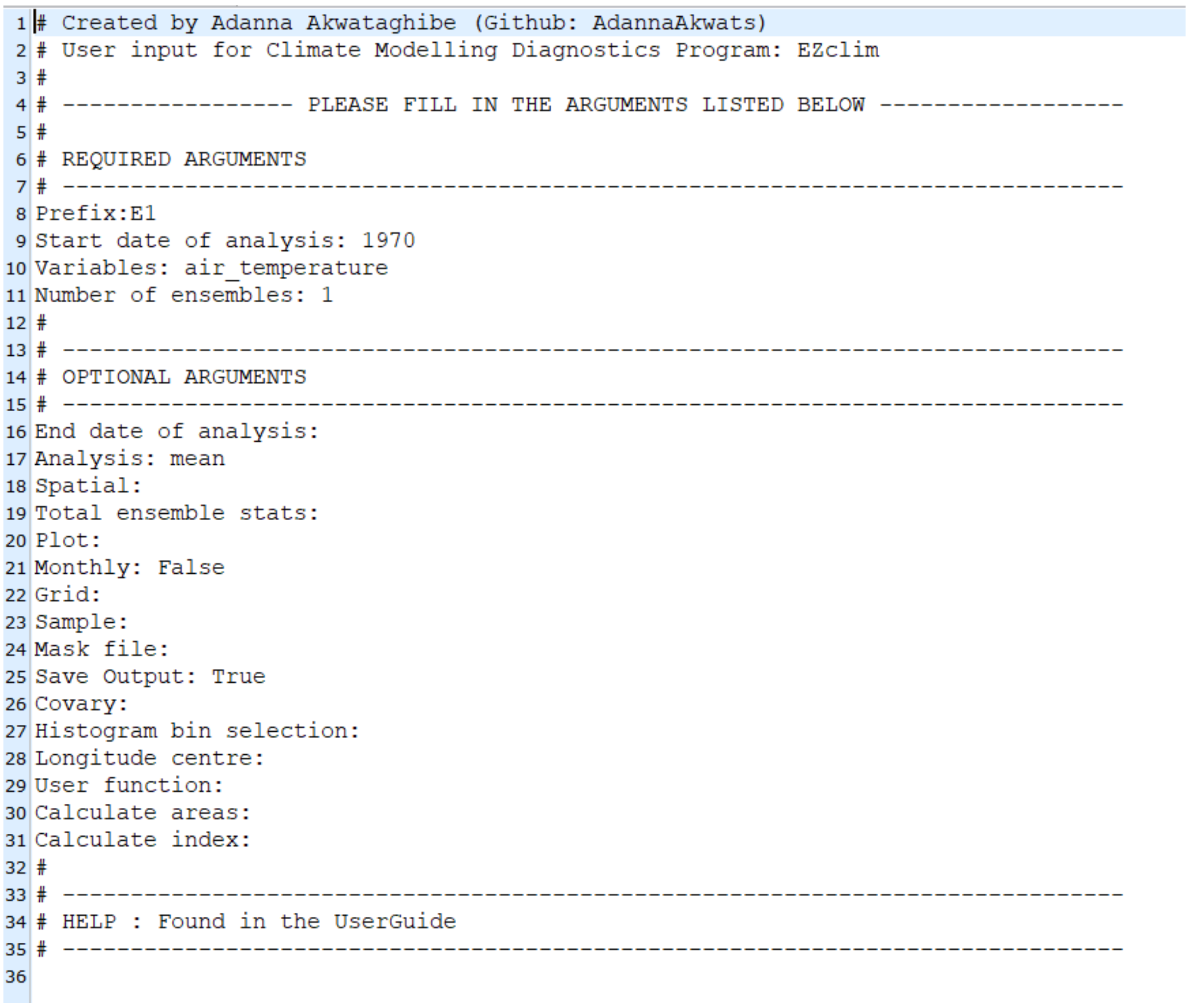
```
                so:comment = "mo: This variable is reported using a z* coordinate system rather than the z-level system indicated by th
ere this is important to analysis please consider the information in time-varying variables thkcello or zfullo., CMIP_table_comment: Se
ty is the salt content of sea water, often on the Practical Salinity Scale of 1978. However, the unqualified term \'salinity\' is gener
t necessarily imply any particular method of calculation. The units of salinity are dimensionless and the units attribute should normal
 1e-3 or 0.001 i.e. parts per thousand." ;
                so:original_name = "mo: (variable_name: so)" ;
                so:original_units = "1e-3" ;
                so:history = "2019-12-05T11:55:05Z altered by CMOR: Converted units from \'1e-3\' to \'0.001\'." ;
                so:cell_methods = "area: mean where sea time: mean" ;
                so:cell_measures = "area: area" ;
                so:missing_value = 1.e+20f ;
                so:_FillValue = 1.e+20f ;
                so:units = "K" ;
        double time(time) ;
                time:bounds = "time_bnds" ;
                time:units = "days since 1850-01-01" ;
                time:calendar = "360_day" ;
                time:axis = "T" ;
                time:long_name = "time" ;
                time:standard_name = "time" ;
        double time_bnds(time, bnds) ;
        float vertices_latitude(lat, lon, vertices) ;
                vertices_latitude:units = "degrees_north" ;
                vertices_latitude:missing_value = 1.e+20f ;
                vertices_latitude:_FillValue = 1.e+20f ;
                vertices_latitude:cell_measures = "area: area" ;
        float vertices_longitude(lat, lon, vertices) ;
                vertices_longitude:units = "degrees_east" ;
                vertices_longitude:missing_value = 1.e+20f ;
                vertices_longitude:_FillValue = 1.e+20f ;
                vertices_longitude:cell_measures = "area: area" ;
```

As can be seen from origin to the new version. The unit is changed from 0.01 to k and the grid is changed from lat(i , j), lon(i , j), so(time , level , i ,j) to lat, lon, so(time , level , lat , lon).

1.3 Run Diagnostics

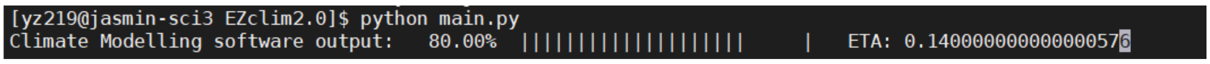
Modify the input.txt file

GitHub repository: https://github.com/acse-2019/irp-acse-yz219

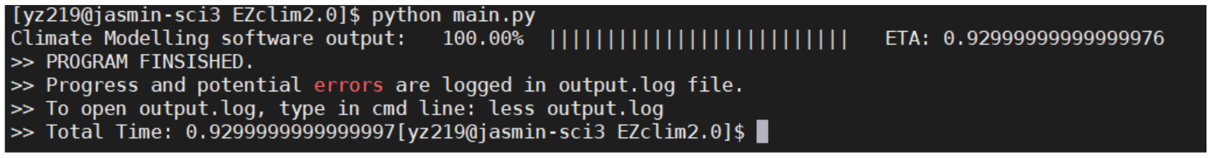


```
# Created by Adanna Akwataghibe (Github: AdannaAkwats)
# User input for Climate Modelling Diagnostics Program: EZclim
#
# ----------------- PLEASE FILL IN THE ARGUMENTS LISTED BELOW ------------------
#
# REQUIRED ARGUMENTS
# -------------------------------------------------------------------------------
Prefix:E1
Start date of analysis: 1970
Variables: air_temperature
Number of ensembles: 1
#
# -------------------------------------------------------------------------------
# OPTIONAL ARGUMENTS
# -------------------------------------------------------------------------------
End date of analysis:
Analysis: mean
Spatial:
Total ensemble stats:
Plot:
Monthly: False
Grid:
Sample:
Mask file:
Save Output: True
Covary:
Histogram bin selection:
Longitude centre:
User function:
Calculate areas:
Calculate index:
#
# -------------------------------------------------------------------------------
# HELP : Found in the UserGuide
# -------------------------------------------------------------------------------

```

Run python main.py



```
[yz219@jasmin-sci3 EZclim2.0]$ python main.py
Climate Modelling software output:   80.00%  ||||||||||||||||||||     |   ETA: 0.14000000000000057
```



```
[yz219@jasmin-sci3 EZclim2.0]$ python main.py
Climate Modelling software output:   100.00%  ||||||||||||||||||||||||||   ETA: 0.9299999999999976
>> PROGRAM FINSISHED.
>> Progress and potential errors are logged in output.log file.
>> To open output.log, type in cmd line: less output.log
>> Total Time: 0.9299999999999997[yz219@jasmin-sci3 EZclim2.0]$
```