The *Atlas of Variability* Code Package and How to Use It

Kevin Schwarzwald

November 26, 2017

Contents

0. Introduction

The *Atlas of Variability* Code Package was developed as part of the "Atlas of Variability" climate statistics project and is designed to allow for a quick, standardized analysis of the variability characteristics of climate model output. It is a series of Matlab functions and auxiliary programs that take NetCDF files and spit out results from spectral density analysis and basic distribution characteristics, and present them in several useful visualizations.

Throughout this guide, `code input`, `function names` and `filenames` will be written in equal-spaced font, while **variables** will be written in bold.

The code is written in MATLAB, and is based on the creation of standardized `.mat` files from the Raw NetCDF data. This procedure allows added code simplicity at the price of slower loading speeds, and it is suggested that future versions of this code allow support for keeping data in NetCDF format.

Output is generally in `.mat` format for data and `.eps` and `.png` format for images. These are defaults that are simple to change (in the code itself) if desired.

The code package assumes one major set of calculations for each variable-data frequency-model-experiment(-run) combination. This procedure was sufficient for the large inter-model comparison done for the "Atlas of Variability," but may cause problems if analyses of the same time series in multiple temporal chunks or analyses of different runs of the same individual model are desired. The code package is slowly being updated to allow for a more robust treatment of at least multiple runs per model, but a better treatment of different temporal chunks would require a change in filename standards for maps and ratio data (the pure output of the variability calculations is largely unaffected, since it uses the full original filename convention with only simplifying the dates into years). This is of course not an insurmountable obstacle by any means, and the user is encouraged to go down that path if the code package has not been sufficiently generalized at the time of use.

Furthermore, the code makes several assumptions and simplifications. Generally, the default settings of the code assume calendars with 365-day years, and the pre-processing functions accordingly remove the 366th day of leap years to ensure compliance. This does not mean that the rest of the code cannot handle calendars with leap years or 360 days etc., but rather defaults must likely be changed or at least carefully considered to ensure the results are exactly what are desired (for example, one of the default 'frequency bands' for daily data spans 30 - 365 days, which makes climatological sense in a 365-day calendar, but not in a 360-day calendar). Additionally, the code

package is primarily set up to deal with time series chunks beginning on January 1st and ending on December 31st of any year. Though sub-annual (seasonal) analysis tools are included, the pre-processed time series, for filename simplicity, are expected to span integer multiples of years.

Finally, filename conventions are based on CMIP5 defaults. The pre-processing code expects raw NetCDF files of a certain filename format, and does calculations based on values in certain positions in those filenames. See more details below.

In general, users of this code package are expected to know their way around MATLAB (i.e. "do you know what the differences between a cell array, a struct, and a numerical array are?') and have a basic understanding of the NetCDF file structure. Every function has a detailed header description that often includes use examples and can be accessed by typing `help [function name]` into Matlab.

For any questions, comments, concerns, complaints, etc., feel free to contact the author (Kevin Schwarzwald, kschwarzwald@uchicago.edu).

Figure 1: Function List (excluding third-party functions

Function Name	Type	Description
various_defaults	Aux	Set file saving and processing defaults
Saves	Pre-processing	Convert NetCDF files into .mat files
var_chars	Aux	Access variable-frequency information
name_chars	Aux	Access individual file information
File_Domain	Aux	Sets file to save figures in
Variability	Variability	Calculate frequency band contributions to variability
Variability_seasons	Variability	Calculate intra-/inter-seasonal variability
Variability_ci	Variability	Get confidence intervals on variability changes
Spectral_Ratios	Aux	Calculate ratio of variability or summary statistics
load_stddevs	Aux	Load files, file subsets, and characteristics of variability data
load_means	Aux	Load files, file subsets, and characteristics of mean data
Maps_ratio	Mapping	Map changes in variability
Maps_diagnostics	Mapping	Map diagnostics/summary statistics of a variable
data_colormap	Aux	Assign colors to array elements
clip_lat	Aux	Get indices of pixels within a latitude range
clip_region	Aux	Get indices of pixels within a geographic region
clip_ci	Aux	Get indices of pixels with non-significant variability changes
clip_minval	Aux	Get indices of pixels above a certain value
clip_maxval	Aux	Get indices of pixels below a certain value
subset_find	Aux	Find indices of one array within another
subplot_pos	Aux	A collection of figure-friendly subplot positions
loc_descriptor	Aux	Get human-readable political description of a lat/lon pair
quote_me	Aux	Just spits out a positive quote about science

1 Raw Data Processing

1.1 Setting Processing and Saving Defaults

All saving and processing defaults are stored in the file `various_defaults.mat` and set by running the function `various_defaults.m`. This program and file controls the directories for storing code, raw data, processed data, seasonal data, figures, and grid area weights. These must be set to the desired paths before running any other function in this package. These directories do not necessarily need to be different from another. NOTE: the code assumes that data is stored in model-specific subfolders of the relevant directories set by `various_defaults.mat`. For example, raw NetCDF data for CCSM4 should be saved in the directory `[raw_data_dir]/CCSM4/`, with `[raw_data_dir]` set by `various_defaults.m`. Please check the function header itself for a full list of which directories imply subdirectories.

`various_defaults.mat` also sets the default frequency bands calculated by `Variability.m` (see Section 3.1), `Variability_ci.m` (see Section **??**), and `Variability_seasons.m` (see Section **??**). Finally, `various_defaults.mat` sets model and experiment name conversions for display purposes (the filename versions of many experiment and model name are different from the human-readable names) and some auxiliary defaults useful for annotations. See the header of the file (`help various_defaults` with it on the path) for more detailed descriptions.

1.2 File Convention

All functions beyond the pre-processing / saving functions assume the raw data is stored in a specific standardized (`.mat`) format in the `[raw_data_dir]/[model_name]/` set by `various_defaults.mat` run above, with the filenaming convention:

$$\begin{aligned}\texttt{filename} = [\text{file variable identifier}]\_[\text{data frequency}]\_[\text{model}]\_[\text{experiment}]\_ \\ [\text{run}]\_[\text{start year}]\text{-}[\text{end year}]\_\text{DATA.mat}\end{aligned} \tag{1}$$

file variable identifier shorthand variable name, e.g. `pr` for precipitation

data frequency month, day, etc., using CMIP5 conventions (_Amon/_Lmon/_Omon for month, for whatever reason)

model model name in UNIX-safe character convention (so CSIRO Mk3.6.0 becomes CSIRO-Mk3-6-0)

experiment experiment / forcing profile, e.g. RCP8.5

run run ID (in CMIP5, [r#i#p#] for run number, iteration, and microphysics; for non-CMIP5, can be anything)

start year-end year e.g. 2070-2099

and the following component parts:

lat latitude, either in vector [$m \times 1$ `double`] or array / grid ([$n \times m$ `double`] array) format

lon longitude, either in vector [$n \times 1$ `double`] or array / grid ([$n \times m$ `double`] array) format

lat_bnds (optional) latitude bounds, with one additional dimension of size 2 compared to the latitude variable, giving the latitude vertices of each pixel

lon_bnds (optional) longitude bounds, with one additional dimension of size 2 compared to the longitude variable, giving the longitude vertices of each pixel

Raw [$n \times m \times t$ `double`] array of data for one variable-frequency-model-experiment-run-timeframe combination, for number of longitude steps n, latitude steps m, and time steps t. **Raw** should always be 3-dimensional. 4-dimensional variables (such as pressure-delimited wind speed) should be split up by level in the 4th dimension into individual 3-dimensional files.

The file naming convention is based on the conventions developed for the CMIP5 project, as are the variable and unit conventions.

In general, the filename format above is used for most *calculated* data files as well, with only the suffix changed (for example, a file with variability output would substitute `_sqrtPower.mat` for `_DATA.mat`).

1.2.1 File Format

In general, files should be stored as version 7.3 (HDF5, partial variable loading possible) with version 6 files (for compatibility with programs such as *R* - the package *R.matlab* can't deal with HDF5 files) saved separately if needed, and with the suffix `_DATA_v6.mat`.

1.2.2 File Saving Order

Files should generally be saved in ascending order of variable size, meaning almost always **lat**, **lon**, (**lat_bnds**, **lon_bnds**,) and **Raw**. This format allows for quicker loading of the geographic grid variables due to the way MATLAB compresses files in their newest file formats (accessing later variables requires some decompression of the previous variables).

1.2.3 File Variable Identifiers

For most variables, the file variable identifier should be modeled on the CMIP5 defaults (see the CMIP5 standard output (link)). The most notable exception is 4-dimensional variables, such as pressure-delimited cloud or wind cover (**cl** and **ua**/**va** in CMIP5 format). In these cases, it is suggested to find a descriptive level-based identifier, for example **cllow** for low cloud cover or **ua850** for 850mb zonal wind velocity.

1.3 Saving Procedure

The *Atlas of Variability* Code Package includes one program designed to process CMIP5 NetCDF data into the `_DATA.mat` format, `Saves.m`. Given a `[var_idx]` representing a variable and frequency in addition to a model, `Saves.m` looks into the `[raw_data_dir]` from `various_defaults.m` to find one or more NetCDF files, and collates/clips them into a `.mat` file in the above convention of the desired length (by default, 30 years; see the function header for options on changing what chunk is extracted and saved).

Any data that follows the CMIP5 naming convention and the required NetCDF file format should be processable without any modification of `Saves.m`, with built-in options for setting the

variable and attribute names to look for in the `.nc` file when they are different than the CMIP5 format included in the program. See the `Saves.m` header (by calling `help Saves`) for full details.

Daily / sub-daily data should generally be stored *without leap years*! `Saves.m` does this automatically by removing the 366th day of leap years in gregorian / julian / proleptic gregorian calendars.

2 General Function Form

Most non-auxiliary functions in this code package use a similar input concept. In general, these functions are called

$$([\text{output}]) = \texttt{function}([\text{var\_idx}], [\text{model}], \dots) \tag{2}$$

var_idx a row index or vector of row indices corresponding to a specific variable - data frequency combination(s) in the file `Varnames.csv` (see below)

model model name or cell array of model names in UNIX-safe character convention (so CSIRO-Mk3.6.0 becomes CSIRO-Mk3-6-0)

2.1 Varnames.csv and How to Set Variable Name Conventions

The code package receives information about variable makeup from the text file `Varnames.csv`. Each row is one variable - data frequency combination (e.g. `pr_day_` ≡ daily precipitation). Different programs in the code package will treat data differently based on the frequency (whether to remove leap years, how to calculate confidence intervals, etc.). The rest of the columns are purely used in graphing and displays. In accessing a row in the document, functions access standardized long names, units, and data types (ratio, interval, temp, etc., for choosing appropriate colorscales and colorbars in plotting) in addition to the shorthands for variable name and frequency used in filenames. This information is accessed in the code package using the function `var_chars.m`. .

The **var_idx** mentioned above and called in most programs is simply the row index of the `Varnames.csv` file corresponding to the desired combination of variable and frequency. If desired, this procedure can be used to also differentiate between any of the graphing/display columns (i.e. have two columns for daily precipitation, but one with SI units (kgm^2/s) and one with meteorological units (like cm/s)). In this case, inserting the different row indices as **var_idx** in programs will ensure that the resultant output displays the desired units.

Support for variables can be added by simply adding a row, and information can be changed by editing the relevant element. Every row should include the information below (by column header):

filevar CMIP5 variable shorthand

filevarFN database variable shorthand (may be different from CMIP5, for example **ua850** for 850mb zonal wind velocity)

freq data frequency, following the CMIP5 standard (which means _Amon, _Lmon, etc.) and adding underscores before and after for filename processing purposes

vardesc variable long name, used in figure captions, etc.

units data and standard deviation units, surrounded by parentheses and a space (to counteract a graphics glitch in MATLAB text renderers)

varunits variance units (data units squared), in the same format

clim_type data type deciding colorbar limits and colors - either **ratio** (greyscale), **interval** (blue to white/grey to red centered at 0), **percent** (greyscale, limits of [0, 100]), or **temp** (blue to white/grey to red, centered at 273)

id row number for easier reference

Though much of this data is theoretically accessible through the original NetCDF files, the idea behind a standardized variable 'dictionary' is to make sure variable names and units are kept constant between climate projects that may use different conventions for all of these.

If a future version of this code package were to switch entirely to keeping data as NetCDF files, the information contained in `Varnames.csv` could be saved in the NetCDF header, with [**variable ID #**] being replaced in program input by a variable name and data frequency combination.

2.2 Common Function Flags and Input Conventions

Though the reader is encouraged to refer to the header of each function for an exhaustive list of function flags and program run modifications (by typing `help [function name]` into the MATLAB console), a few flags are general across the code package and are worth enumerating here:

'testing' adds the suffix `_TEST` to the output filename(s)

'experiments' sets the experiment runs processed, in cell array format (e.g. `{rcp85,piControl}`, must be written as in filenames; different display names, for example RCP8.5 instead of rcp85, are supported through changing the **expArray_disp** array in `various_defaults.m`)). Some code, such as `Maps_ratio.m` or `SpectralRatios.m`, requires multiple experiment inputs in a specific order, please see function headers for exact requirements.

3 Calculating Variability

3.1 Calculating Frequency Contributions to Variability

The core purpose of this program package is to calculate the frequency-separated variability of an arbitrary climatic variable; in other words, the portion of the standard deviation contributed to by patterns within certain frequencies. This is done through the following process (a more rigorous description is found in Appendix A):

1. Detrending: a simple linear trend is removed from the time series (spectral density analysis requires stationarity)
2. Deseasonalization: the seasonal component of variability is removed by subtracting out a 12-harmonic least-squares fit
3. Spectral Density Calculation: a power spectrum is calculated from the resultant time series
4. Spectral Density Integration: the standard deviation is calculated by taking the square root of the integrated spectral density between the desired frequencies

This process is executed through the function `Variability.m` (see the function header for a complete description). `Variability.m` takes in a `_DATA.mat` file and outputs two files, with suffixes `_LocalMeans.mat` and `_sqrtPower.mat`. The former simply contains **lat**, **lon**, and a variable **LocalMeans**, an $[n \times m]$ array of the mean value of each grid point in the `_DATA.mat` file over the relevant time period. The latter contains **lat**, **lon**, and a struct **StdDevs** that contains the calculated variability data in addition to the mean data already saved in `_LocalMeans.mat`. The structure of **StdDevs** is as follows:

StdDevs.Name {`'< 5 days','5 - 30 days',...,'all frequenices'`} (long name of frequency band, used in descriptions and graph titles)

StdDevs.ID {`'HF','MF',...,'Full'`} (short identifier of frequency band, used in function input)

StdDevs.Size {`[0,5],[5,30],...,[0 Inf]`} (period limits of each frequency band, for reference)

StdDevs.Data {$[n \times m$ `double],...`} (standard deviation portion contributed by the frequency band)

The struct **StdDevs** also includes entries for the local mean values (as in `_LocalMeans.mat` above) stored with **ID** = `'mean'` and the standard deviation stored with **ID** = `'std'`. The standard deviation is usually not exactly equal to the `'Full'` / all frequency band due to estimation precision in the spectral density calculation. Both `'std'` and `'mean'` are stored with **Size** = `[ ]` (an empty array).

This and the following two variability-calculating functions are optimized for parallel computing and by default parallelize processing across 12 processors. This can be changed in each function call.

3.2 Calculating Seasonal Variability

Intra- and inter-seasonal variability can be calculated through `Variability_seasons.m`. As with `Variability.m`, the function `Variability_seasons.m` takes in a `_DATA.mat` file and outputs two files containing the local means and a struct with variability and summary statistics data, though now with the addition of a seasonal suffix: `_LocalMeans_[seas].mat` and `_StdDevs_[seas].mat`, respectively. The suffix `[seas]` is set by default in `various_defaults.mat` and is generally a shorthand of the included months (i.e. `DJF` for Dec-Jan-Feb). By default, `DJF` and `JJA` are processed; for other seasons, please see the header of the function.

The frequency bounds for intra-seasonal variability calculations are taken as before from `various_defaults.mat` (the variable **freq_bands_setup_seasons**), with customizable bounds set in the function call (see the header). Interseasonal variability ($>$ `1 year`) is defined as the standard deviation of seasonal means across years in the data set.

By default, the function processes 30 seasons. If the start and end dates of a season are in different calendar years, the function will attempt to take data from 31 (or `[nyears] + 1` for custom year lengths) years of raw data to keep the total number of seasons contained in the analysis consistent. Warnings will be thrown if this is not possible.

3.3 Calculating Variability Ratios

For a slight simplification in manual examination of variability change data, the code package allows for the creation of extra files that store the ratio of frequency-separated variability between two

experiments (these files are also necessary to run `Maps_ratio.m`). This process occurs through the program `SpectralRatios.m`. The function is called as normal, with an important input to remember:

$$\texttt{SpectralRatios}([\text{variable ID \#}], [\text{model}], (\text{`experiments'}, \{exp_2, exp_1\}), \dots) \tag{3}$$

exp_2 and exp_1 are the two experiments between which the ratio should be calculated. Specifically, the ratio will be calculated as $\frac{exp_2}{exp_1}$ and is by default set as $exp_2 =$ `rcp85` and $exp_1 =$ `piControl`.

The output file has the suffix `_SpectralRatios.m` and the same structure as the `_sqrtPower.m` file explained above, with the only difference being the name of the struct (**StdDevsRatio** instead of **StdDevs**) and the addition of a variable **file_params** with the experiments, runs, and start/end years of the `_DATA.mat` files used in the program run. Importantly, the output filename no longer has the [`year`] field - in the current structure of the file system only one ratio file can be stored per run and experiment combination. To ensure that the SpectralRatios file is created from time series with the same length, the program cycles through `_sqrtPower.m` files to find a matching pair (with warnings being thrown if equal length time series could not be found for both experiments).

3.4 Calculating Confidence Intervals on Variability Changes

`Variability_CI.m` allows the generation of confidence intervals on ratios of frequency-separated variability (and any other quantity saved in the **StdDevs** structs, such as mean or standard deviation). In absence of a large ensemble of runs of the same model, this is done using a boostrap methodology (see Appendix A for derivation). NOTE: currently this procedure only supports daily data! Similar to `SpectralRatios.m`, `Variability_CI.m` requires the input of two experiments:

$$\texttt{Variability\_CI}([\text{variable ID \#}], [\text{model}], (\text{`experiments'}, \{exp_2, exp_1\}), \dots) \tag{4}$$

The output file has the suffix `_StdDevCI.mat` and has the same variables as `_StdDevsRatio.mat` files above (**lat**, **lon**, **StdDevsRatio**) with the addition of a variable **file_params** with the experiments, runs, and start/end years of the `_DATA.mat` files used in the program run.

The variable **StdDevsRatio** has the fields (the first four are equivalent to **StdDevs** and **StdDevsRatio** above):

StdDevs.Name {`'< 5 days','5 - 30 days',...,'all frequencies'`} (long name of frequency band, used in descriptions and graph titles)

StdDevs.ID {`'HF','MF',...,'Full'`} (short identifier of frequency band, used in function input)

StdDevs.Size {`[0,5],[5,30],...,[0 Inf]`} (period limits of each frequency band, for reference)

StdDevs.Data {[$n \times m$ `double],...`} (standard deviation portion contributed by the frequency band)

std_runs {[$n \times m$ `double],...`} the local standard deviation σ across bootstrap runs of the ratio of variabilities; confidence intervals are determined as 2σ from the calculated ratio

lon_ciidx {[$k \times 1$ `double],...`} longitude indices of pixels in which the ratio of variabilities is not meaningfully different from 1 (the confidence interval includes 1)

lat_ciidx {[$k \times 1$ `double],...`} latitude indices of pixels in which the ratio of variabilities is not meaningfully different from 1 (the confidence interval includes 1)

4 Mapping Results

4.1 Mapping ratios of variability changes

The code package includes some basic mapping tools. The most robust, and most used in the "Atlas of Variability" Project is `Maps_ratio.m`, which produces a map / up to 6 subplotted maps of the change in variability between two experiments of a variable - frequency - model combination, and can hash out pixels without a meaningful change in variability, as calculated in `Variability_CI.m`. It requires a previously calculated `_SpectralRatios.mat` file for the experiments to be compared, and, if a marking of pixels without meaningful changes is desired, a previously calculated `_StdDevsCI.mat` file.

Beyond the general functional form (as always, check the function header through `help Maps_ratio` for all flags and details), I would like to point out two specific function flags:

'show_freqs' a cell array of the form {`'HF','MF',...,'Full'`} (with 1 - 6 cells), including the **ID**/short data identifier of the desired frequency band / quantity to be graphed

'plot_method' followed by either `'pcolorm'` or `'polygon'`. The former is the default, and uses the matlab function `pcolorm` to produce a geographic checkboard/pseudocolor plot. However, the pcolorm paradigm plots data at vertices, and so removes a row and column from each side (see here for details). The option `'polygon'` instead graphs each individual pixel as a filled polygon, solving this issue at the expense of computational efficiency (plotting and saving individual polygons results in a significant increase in processing time, and is therefore generally suggested to be used for production figures only). An added benefit of the `'polgyon'` option is the ability to more accurately show the shape of pixels in non-rectangular grids. Pixel edges are rectangular and taken from the **lat_bnds** and **lon_bnds** variables in a NetCDF file from the same model with the same lat/lon grid as the data. In other words, it searches the raw data folder's (set in `various_defaults.m`) relevant model subset directory for a NetCDF file with the same lat/lon grid and the variables **lat_bnds** and **lon_bnds**. If this is not found, latitude and longitude bounds are inferred as being halfway between pixel coordinates, and a warning is thrown.

4.2 Mapping some summary statistics

In addition to `Maps_ratio.m`, this package also includes the mapping function `Maps_diagnostics.m`, which creates a set of 4-panel maps showing 1) mean values for exp_1, 2) exp_1 standard deviations, 3) difference in mean values exp_2 - exp_1, and 4) ratio of standard deviations $\frac{exp_2}{exp_1}$for a variable - model combination. By default, exp_1 and exp_2 are `piControl` and `rcp85`, respectively. See the function header for more detail.

`Maps_diagnostics.m` requires color limits across models that it takes from running `MapLimits.m`. `MapLimits.m` creates a file with intermodel statistics that allow for a consistent intermodel colormap creation (these are *not* rigorous summary statistics). See the function header for more detail.

5 Graphing Results

Explicit graphing functions (beyond the mapping functions detailed above) are not fully supported in this code package (a beta scatterplotting function, `panel_plots.m` is included; see its header

for a description and use at your own risk); however, certain auxiliary tools are included to make graphing easier.

5.1 Subsetting data by region, value, etc.

The following are several functions designed to give linear indices of pixels that fulfill a certain geographic, numerical, or meaningfulness criteria. In general, these `clip_*.m` files follow a basic input/output paradigm:

$$\begin{aligned} \textbf{idx\_clipped} &= \texttt{clip\_*(...)} \\ \textbf{idx\_unclipped} &= \texttt{clip\_*(...,idxs,...)} \end{aligned} \tag{5}$$

These two forms allow for both getting the linear indices of pixels that match the criteria determined through `clip_*` (**idx_clipped**) or the linear indices of pixels that do not match the same criteria (**idx_unclipped**). In the latter case, an existing vector of 'unclipped' indices **idxs** should be added (can be an empty vector, see individual function headers for a more detailed description).

5.1.1 Clipping data by region

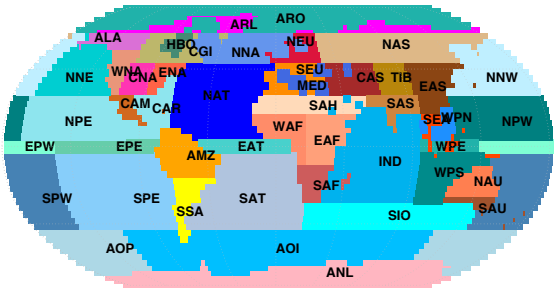

Figure 2: Map of regions used in `clip_region.m`

`clip_region.m` allows the return of (linear) indices of an array of points lying within a geographic region or multiple regions. The basic function call is:

$$\textbf{idx\_clipped} = \texttt{clip\_region(region\_id,model,lat,lon)} \tag{6}$$

The information needed to set region boundaries is found in the file `regions.mat` in the basic code folder. This file contains a struct **Regions**, containing a lat/lon bounding box (the fields **lon_min**, **lon_max**,**lat_min**, **lat_max**), whether the region is land or ocean (the field **Type** = `'LAND'`, `'OCEAN'`, or `'LAND'`), and a short ID and long name for the region (the fields **ID** and **Name**). Each 'region' is defined as points lying within a lat/lon bounding box, and potentially limited to just land or ocean pixels. More regions can be added by editing the struct **Regions**. The existing regions are those used by [1] (see Figure 2).

The lat/lon grid does *not* have to be global. Indices will be returned based on the inputted lat/lon grid.

region_id in the function input above can either be the field **ID** of the relevant region or its index in the struct **Regions**. Multiple regions can also be inputted simultaneously in a cell array {`region_id`$_1$`,region_id`$_2$`,...`}. In this case the resultant output is based on the union of pixels in the inputted regions.

The model needs to be inputted (in normal string format, i.e. `'CSIRO-Mk3-6-0'`) because of the need for a surface-land fraction grid to determine land/ocean pixels in a specific model. If at least one of the desired regions is land or ocean only, the program then looks in the raw data folder for the model for a file `sftlf*.mat` with a matching lat/lon grid as inputted in the variables **lat** and **lon**. No such action if the region(s) inputted are not land- or ocean-specific (and the `sftlf*.mat` file is then not needed). The latitude and longitude grid does not have to cover the whole globe! As long as the lat/lon grid can be found within the `sftlf*.mat` grid, the correct indices will be output, with an error if the subset was not found.

5.1.2 Clipping data by latitude band

`clip_lat.m` allows the return of (linear) indices of an array of points lying between two latitude values (absolute, by default). The basic function call is:

$$\textbf{idx\_clipped} = \texttt{clip\_lat(lat\_min,lat\_max,lat,lon)} \tag{7}$$

By default, the indices of points are given that lie between the absolute value of `lat_min` and `lat_max`; in other words, for both the Northern and Southern hemispheres. To just return indices in a specific hemisphere, the flag

$$\textbf{idx\_clipped} = \texttt{clip\_lat(...,'hemisphere',['NH'/'SH'],...)} \tag{8}$$

can be used. As an additional output, the exact latitudes used can be shown in a struct, see the specific function documentation for more information.

5.1.3 Clipping variability data by significance

`clip_ci.m` allows the return of (linear) indices of points determined to have an insignificant change in the variable between experiments, as calculated through `Variability_CI.m` (see Section 3.4). The basic function call is:

$$\textbf{idx\_clipped} = \texttt{clip\_ci(var\_idx,model,expArray)} \tag{9}$$

This clip requires a relevant `_StdDevCI.mat` file calculated through `Variability_CI.m` to function.

6 Auxiliary Functions

6.1 Loading calculated data

The filenames used in the CMIP5 models and throughout this code package are admittedly a bit of a mouthful. Here's a few functions designed to make loading data a bit simpler and more pleasant.

6.1.1 Loading data from _sqrtPower.mat files

`load_stddevs.m` allows for the direct loading of the struct **StdDevs**, the lat/lon grid, or even subsets of the **StdDevs** struct (such as the IDs contained within it, etc.). The basic function call is as follows:

$$\textbf{[StdDevs,lat,lon]} = \texttt{load\_stddevs([var\_idx],[model],[experiment],([season]),...)} \tag{10}$$

To make loading such data even easier, `[var_idx]` can be replaced by the CMIP5 shorthands for variable and frequency, for example: `load_stddevs('pr','day',...)`. To get the filename that the function is loading (to verify timeframe, etc.), add the flag `'filename'`. To get properties within the **StdDevs** struct, use the flag `'property',[field name]`.

6.1.2 Loading data from _LocalMeans.mat files

`load_means.m` allows for the direct loading of the file LocalMeans or the lat/lon grid. The basic function call is as follows:

$$\textbf{[LocalMeans,lat,lon]} = \texttt{load\_means([var\_idx],[model],[experiment],([season]),...)} \tag{11}$$

The data in **LocalMeans** is also found as **ID** = `mean` in **StdDevs** above, but this allows a more direct loading of the mean values of a time series.

6.2 Getting subsets of arrays

`subset_find.m` allows for finding the indices of subsets of arrays within larger arrays. The basic call is as follows:

$$\textbf{[x\_idxs,y\_idxs]} = \texttt{subset\_find}(\textbf{x},\textbf{y},\textbf{x'},\textbf{y'}) \tag{12}$$

where the matrix (**x',y'**) is a subset of the matrix (**x,y**), and **x_idxs,y_idxs** are the indices of each point in (**x',y'**)'s position in (**x,y**). More specifically, in terms of climate data, this procedure allows finding a geographic subset inside a lat/lon grid, through the (identical) call

$$\textbf{[lat\_idxs,lon\_idxs]} = \texttt{subset\_find}(\textbf{lat},\textbf{lon},\textbf{lat'},\textbf{lon'}) \tag{13}$$

6.3 Getting a human-readable location description

`loc_descriptor.m` uses the Google Maps database to return a human-readable description of a lat/lon location. By default,

$$\textbf{loc\_desc} = \texttt{loc\_descriptor(lat,lon)} \tag{14}$$

returns the most complete form of the location's address stored in the Google Maps database. For more comp[ex reverse geocoding (such as just getting the political division using the `'result_type', 'political'` flag) requires your own API key (see function header), which can be inputted using the `'api_key'` flag.

7 Third Party/Community Functions

A few functions (listed in Figure 3) are taken from the MATLAB community.

Figure 3: Third-Party Function List

Function Name	Type	Description	Author
`ds2nfu`	Plotting	Converts from data units to normalized figure units for placing objects onto plots	Michelle Hirsch
`piecelin`	Calculation	Used for piecewise interpolation of colormaps in `data_colormaps.m`	
`structfind`	Aux	Gets index of a struct based on matching a specific field	Dirk-Jan Kroon
`rgb`	Plotting	Returns RGB triplet for CSS3 colors	Kristjan Jonasson
`wraptext`	Aux	Wraps output text in console	Chad Greene

8 Typical Code Run

The following shows an example code run from start to finish analyzing the relationship between changes in daily precipitation variability and mean in the model CSIRO Mk3.6.0 between RCP8.5 and pre-industrial control and between RCP8.5 and historical runs. The `'experiments'` calls aren't strictly necessary (except for RCP8.5 - historical comparisons) since they just represent the default function behavior, but are included for reference.

1. set raw / processed / output data directories in `various_defaults.m` (Section 1.1); add the model to the **modelArray_disp** variable to accommodate the model name ('CSIRO-Mk3-6-0' in the filesystem, 'CSIRO Mk3.6.0' in figure captions); add the experiments to the **expArray_disp** variable to accommodate experiment names ('rcp85' / 'historical' / 'piControl' in the filesystem, 'RCP8.5' / 'Historical' / 'PI Control' in figure captions)

2. download `pr_day_CSIRO-Mk3-6-0_rcp85*.nc`, `pr_day_CSIRO-Mk3-6-0_historical*.nc`, and `pr_day_CSIRO-Mk3-6-0_piControl*.nc` files from ESGF, DKRZ, or your other favorite climate model data repository into the raw data directory set above

3. run `Saves(5,{'CSIRO-Mk3-6-0'},'experiments',{'rcp85','historical','piControl'})` for daily precipitation row index 5 (from `Varnames.csv`, Section 2.1) to pre-process data into 30-year `*_DATA.mat` files and to remove leap days

4. run `Variability(5,{'CSIRO-Mk3-6-0'},'experiments',{'rcp85','historical','piControl'})` (Section 3.1) to calculate integrated spectral densities, means, and standard deviations

5. run `SpectralRatios(5,{'CSIRO-Mk3-6-0'},'experiments',{'rcp85','piControl'})` and `SpectralRatios(5,{'CSIRO-Mk3-6-0'},'experiments',{'rcp85','historical'})` (Section 3.3) to save variability ratios

6. run `Maps_ratio(5,{'CSIRO-Mk3-6-0'}, 'experiments',{'rcp85','piControl'}, 'show_freqs',{'HF','MF','mean','LF'})` and `Maps_ratio(5,{'CSIRO-Mk3-6-0'}, 'experiments',{'rcp85','historical'}, 'show_freqs',{'HF','MF','mean','LF'})` (Section 4.1) to map the < 5 day variability, 5 - 30 day variability, mean, and 30 - 365 day

variability changes in daily precipitation between RCP8.5 and PI control and RCP8.5 and historical runs

7. run `panel_plots(5, {'CSIRO-Mk3-6-0'}, 'experiments', {'rcp85','piControl'}, 'show_freqs', {'LF','XF'}, 'clip_lat',30,55, 'color_by','data', {'piControl'}, {'mean'}, 'log10')` (Section 7)[1] to create a scatterplot of variability changes vs. mean changes for 30 - 365 day and > 1 year variability (in 2 subplots) in the mid-latitudes ($|30° - 55°|$), colored by the base-10 log of pre-industrial mean precipitation

References

[1] S. Castruccio, D. J. McInerney, M. L. Stein, F. L. Crouch, R. L. Jacob, and E. J. Moyer, "Statistical Emulation of Climate Model Projections Based on Precomputed GCM Runs," *Journal of Climate*, vol. 27, pp. 1829–1844, Oct. 2013.

[2] R. Knutti, D. Masson, and A. Gettelman, "Climate model genealogy: Generation CMIP5 and how we got there," *Geophysical Research Letters*, vol. 40, pp. 1194–1199, Mar. 2013.

[3] W. B. Leeds, E. J. Moyer, and M. L. Stein, "Simulation of future climate under changing temporal covariance structures," *Advances in Statistical Climatology, Meteorology and Oceanography*, vol. 1, pp. 1–14, Feb. 2015.

[4] J. M. Klavans, A. Poppick, S. Sun, and E. J. Moyer, "The influence of model resolution on temperature variability," *Climate Dynamics*, pp. 1–11, Aug. 2016.

A Mathematical background and derivations

High- (i.e. the pattern of rain showers) and low-frequency (i.e. droughts, extreme precipitation events) patterns of precipitation are affected by different climatological processes, and could therefore react differently to changes in temperature and gas concentrations in addition to having differing impacts on human society. To isolate changes in variability for different precipitation processes, we use spectral density analysis to study variability in different frequency 'bands' by decomposing the time series for each model output pixel into it frequency components. To do so, we largely adapt the methodology introduced in [3] and [4] based on integrations over power spectra.

Spectral density calculations assume stationarity in the underlying time series, so we first detrend the climatic data. We isolate non-seasonal variability by several data frequency-based deseasonalization methods. Finally, we calculate band-separated variability from power spectra.

A.1 Detrending and Deseasonalization

A.1.1 Daily Data

A classic decomposition of a time series Y is as follows:

$$\begin{aligned} Y &= X_t a + Y_c \\ Y &= m_t + s_t + Y_c \\ Y_c &= Y - m_t - s_t \end{aligned} \tag{15}$$

[1] in beta

for trend m_t, seasonal component s_t (with a known period), and a variability component Y_c. m_t is either 0 if the time series is already stationary (as it is with equilibrated CCSM3 runs) or is made to equal 0 through subtracting a linear trend, leaving

$$Y_c = Y - s_t \tag{16}$$

The seasonal component is then removed by fitting harmonic components using least squares. Given the least-squares process

$$Y_c = Y - s_t\hat{\beta} \tag{17}$$

for

$$\hat{\beta} = \frac{s_t' s_t}{s_t' Y}$$

we fit

$$s_t = \left[1 \cos\left(\frac{2\pi}{365} \begin{bmatrix} 1 \\ \vdots \\ t \end{bmatrix} [1 \dots \lambda] \right) \sin\left(\frac{2\pi}{365} \begin{bmatrix} 1 \\ \vdots \\ t \end{bmatrix} [1 \dots \lambda] \right) \right] \tag{18}$$

for length of (daily) time series t and number of harmonics to be removed λ. In the general process for daily time series used in this project, $\lambda = 12$, and for 30-year time series, $t = 10950$. Subtracting $s_t\hat{\beta}$ from equation 17 above results in Y_c now representing the detrended, deseasonalized, variable component of the time series, ready to be further analyzed.

A.1.2 Monthly Data

Monthly data was deseasonalized by taken the simple average of each month over the length of each time series and subtracting it from every data month. Y_c, the deseasonlized time series Y over T years, was constructed as follows:

$$Y_c(y, m) = Y(y, m) - \frac{1}{T}\sum_{y=1}^{T} Y(y, m) \tag{19}$$

for each month m and year y.

A.2 Spectral Analysis

The autocorrelation function for a stationary process $x(t)$ with mean $\mu = 0$ and variance σ^2 is given by

$$R(\tau) = \frac{1}{\sigma^2} E[(x_t - \mu)(x_{t+\tau} - \mu)]$$
$$R(\tau) = \frac{1}{\sigma^2} E[x_t x_{t+\tau}]$$

and is periodic at the same period as the original function $x(t)$. Peaks in $R(\tau)$ correspond to periodicites with frequencies τ - the autocorrelation function finds interior periodicites in the original time series. By the Wiener-Khinchin Theorem, the autocorrelation function makes a Fourier Pair with the power spectral density $S_{xx}(\omega)$ as follows:

$$S_{xx}(\omega) = \int_{-\infty}^{\infty} R(\tau) e^{-i\omega\tau} d\tau$$

The absolute value of the Fourier Transform as a function of frequency gives the amount of that frequency that is present in the original function, in this case $R(\tau)$. Therefore the (infinite) sum of the Fourier transforms over a range of frequencies gives the contribution of those frequencies to the autocorrelation, which gives how strongly different frequency patterns show up in the time series. In other words,

$$\begin{aligned}\int_{\omega_1}^{\omega_2} S_{xx}(\omega)d\omega &= \frac{1}{T}\int_{\omega_1}^{\omega_2} |\hat{x}(\omega)|^2 d\omega \\ &= \text{contribution to power by } \omega \in [\omega_1, \omega_2]\end{aligned} \tag{20}$$

Now, taking the sample variance of a discrete stationary time series x of length N with mean $\mu = 0$,

$$\sigma^2 \equiv \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i)^2$$

and assuming an infinite time series ($N \to \infty$), we see that the variance of a time series is related to its average power $\bar{P}$ over the domain $[-T, T]$ through

$$\sigma^2 = \lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N}(x_i)^2 \to \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} x(t)^2 dt = \bar{P}$$

Using Parceval's theorem, which states

$$\int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |\hat{x}(\omega)|^2 d\omega$$

for the Fourier transform $\hat{x}(f)$ of $x(t)$, we can see that

$$\frac{1}{2T}\int_{-\infty}^{\infty} |\hat{x}(\omega)|^2 d\omega = \frac{1}{T}\int_{0}^{\infty} |\hat{x}(\omega)|^2 d\omega = \sigma^2$$

Combining this expression with equation 20 above,

$$\frac{1}{T}\int_{\omega_1}^{\omega_2} |\hat{x}(\omega)|^2 d\omega = \sigma^2\{\omega \in [\omega_1, \omega_2]\} \tag{21}$$

with the standard deviation contained in those frequencies simply given by the square root of the expression,

$$\sigma(\vec{\omega}) \equiv \sigma\{\omega \in [\omega_1, \omega_2]\} = \sqrt{\frac{1}{T}\int_{\omega_1}^{\omega_2} |\hat{x}(\omega)|^2 d\omega} \tag{22}$$

where we have used $\sigma(\vec{\omega})$ for ease of notation in the article text.

A.3 Bootstrap Procedure

Given the time series representing precipitation at a single pixel for one experiment run spanning n years, for each bootstrap run a new time series was created also spanning n 'years', but with each year starting at a random time step in the original time series. The years are allowed to 'wrap-around', so if a randomly chosen start interval was less than a year from the end of the original time series, the counter continues at the start of the time series instead. For example, for a 30-year time series, a new time series was created using 30 randomly chosen start indices in the domain [1,10950] (10950 days in 30 years), counting up 365 days from each start index, and concatenating end-to-end.

Variability contributed by different frequency bands were calculated as above for each run. Then, for each run, the ratio of future / past variability for each frequency band was taken. The standard deviation across runs for this ratio was taken for each frequency band. If the outputted final value of that ratio is within two standard deviations as so calculated of 1 (no change), then the change in variability for that pixel is not considered meaningful. 1200 runs are used by default.

B A Few Frequently Used Terms

modelArray a cell array of models (explicitly the 3rd section of each filename, and also the subdirectory where each file is found) used to select files to input into functions

expArray a cell array of experiments (explicitly the 4th section of each filename) used to select files to input into functions

run a shorthand name for the computational run that created a certain file (explicitly the 5th section of each filename)

frequency band a segment in frequency space bounded by two frequencies for which variability is calculated

CMIP5 "Coupled Model Intercomparison Project Phase 5" - a model intercomparison project whose file structure and naming conventions are assumed throughout this code package

log (in a function header) logical 0/`false` or 1/`true`

cell (in a function header) a cell array of strings, numbers, etc. (e.g. {`'rcp85',...`})