NOAA Model Diagnostics Task Force

MDTF Diagnostics Documentation

Release 4.0

Model Diagnostics Task Force

Apr 23, 2024

CONTENTS

1 Getting started **1**
- 1.1 Overview . . . 1
- 1.2 Installation instructions . . . 2
- 1.3 Running the package on your data . . . 10
- 1.4 Running the MDTF-diagnostics package in "multirun" mode . . . 13

2 Developer information **15**
- 2.1 POD requirements . . . 15
- 2.2 Developer installation instructions . . . 18
- 2.3 POD development guidelines . . . 21
- 2.4 POD settings file summary . . . 24
- 2.5 POD coding best practices . . . 30
- 2.6 Git-based development workflow . . . 33
- 2.7 Development Cheatsheet . . . 41

3 Diagnostics reference **45**
- 3.1 Example Diagnostic Documentation . . . 45
- 3.2 Rich Neale's Blocking Diagnostic Documentation . . . 49
- 3.3 Convective Transition Diagnostic Package . . . 52
- 3.4 Moist Static Energy Diagnostic Package . . . 56
- 3.5 Rossby Wave Sources Diagnostic Package . . . 60
- 3.6 EOF of Geopotential Height Diagnostic Module From NCAR . . . 65
- 3.7 Eulerian Storm Track . . . 66
- 3.8 Multi-Case Example Diagnostic Documentation . . . 68
- 3.9 Forcing Feedback Diagnostic Package . . . 69
- 3.10 Mixed layer depth . . . 71
- 3.11 MJO Propagation and Amplitude Diagnostic Package . . . 72
- 3.12 MJO Suite Diagnostic Module From NCAR . . . 76
- 3.13 MJO Teleconnection Diagnostic Package . . . 81
- 3.14 Ocean Surface Latent Heat Flux Diagnostic Documentation . . . 84
- 3.15 Precipitation Buoyancy Diagnostic Package . . . 87
- 3.16 Phase and Amplitude of Precipitation Diurnal Cycle . . . 88
- 3.17 Sea Ice Suite . . . 90
- 3.18 Soil moisture-Evapotranspiration Coupling Diagnostic Package . . . 92
- 3.19 Stratosphere-Troposphere Coupling: Annular Modes . . . 95
- 3.20 Stratosphere-Troposphere Coupling: Eddy Heat Fluxes . . . 98
- 3.21 Stratosphere-Troposphere Coupling: Stratospheric Ozone and Circulation . . . 100
- 3.22 Stratosphere-Troposphere Coupling: QBO and ENSO stratospheric teleconnections . . . 103
- 3.23 Stratosphere-Troposphere Coupling: Stratospheric Polar Vortex Extremes . . . 106
- 3.24 Stratosphere-Troposphere Coupling: Vertical Wave Coupling . . . 109

3.25 TC MSE Variance Budget Analysis . . . 112
3.26 TC Rain Rate Azimuthal Average Documentation . . . 115
3.27 Surface Temperature Extremes and Distribution Shape Package . . . 117
3.28 Top-Heaviness Metric Diagnostic Documentation . . . 126
3.29 Tropical Pacific Sea Level Diagnostic Documentation . . . 127
3.30 Wavenumber Frequency Spectra Diagnostic Module From NCAR . . . 131

4 Framework reference **133**
4.1 Command-line options . . . 133
4.2 Recognized conventions . . . 136
4.3 Working with unimplemented conventions . . . 136
4.4 Model data format . . . 137
4.5 MDTF-diagnostics Environment variables . . . 139
4.6 Running Submodules . . . 141

5 Internal code documentation **143**
5.1 Package code and API documentation . . . 143
5.2 Module index . . . 151

6 Tools documentation **153**
6.1 catalog_builder.py . . . 153
6.2 rename_input_files.py . . . 154

CHAPTER

ONE

GETTING STARTED

1.1 Overview

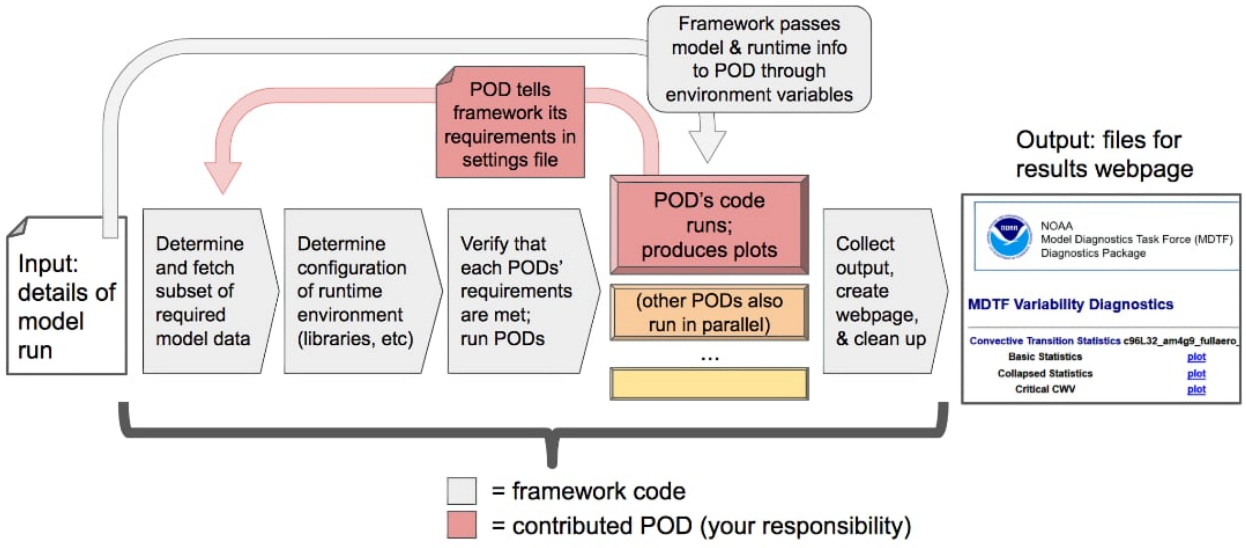
Welcome! In this section we'll describe what the Model Diagnostics Task Force (MDTF) framework is, how it works, and how you can contribute your own diagnostic scripts.

1.1.1 Purpose

The scientific motivation and content behind the framework was described in E. D. Maloney et al. (2019): Process-Oriented Evaluation of Climate and Weather Forecasting Models. *BAMS*, **100** (9), 1665–1686, doi:10.1175/BAMS-D-18-0042.1[1].

1.1.2 Framework operation

The design goal of the MDTF framework is to provide a portable and adaptable means to run process-oriented diagnostic scripts, abbreviated as PODs below. By "portability," we mean the ideal of "run once, run anywhere": the purpose of the framework is to automate retrieval of model data from different local or remote sources, and transform that data into a layout (field names, variable units, etc.) your script expects. This will empower your analysis to be run by a wider range of researchers on a wider range of models.



[1] https://doi.org/10.1175/BAMS-D-18-0042.1

As shown in the figure above, the MDTF framework itself performs common data management and support tasks (gray boxes) before and after the individual POD scripts are run. The PODs (colored boxes) are developed by different research groups and run independently of one another. Each POD takes as input

1. requested variables from the model run, along with
2. any required observational or supporting data, performs an analysis, and produces
3. a set of figures which are presented to the user in a series of .html files.

We do not include or require a mechanism for publishing these webpages on the internet; html is merely used as a convenient way to present a multimedia report to the user.

1.1.3 Getting started for users

The rest of the documentation in this section describes next steps for end users of the framework:

- We provide instructions on how to *download and install* (page 2) the framework and run it on sample model data.
- We describe the most common *configuration options* (page 10) for running the framework on your own model data. Also see the full list of *command-line options* (page 133).
- If you encounter a bug, check the GitHub issue tracker[2].

1.1.4 Getting started for POD developers

Information for researchers wishing to contribute a POD to the framework is provided in the *Developer Information* (page 15) section The framework is designed to require minimal changes to existing analysis scripts. We recommend that developers of new PODs start independently of the framework and adapt it for the framework's use once it's fully debugged. As summarized in the figure above, the changes needed to convert an existing analysis script for use in the framework are:

- Provide a settings file which tells the framework what it needs to do: what languages and libraries your code need to run, and what model data your code takes as input.
- Adapt your code to load data files from locations set in unix shell environment variables (we use this as a language-independent way for the framework to communicate information to the POD).
- Provide a template web page which links to, and briefly describes, the plots generated by the script.

1.2 Installation instructions

This section provides basic directions for downloading, installing and running a test of the Model Diagnostics Task Force (MDTF) package using sample model data. The package has been tested on Linux, Mac OS, and the Windows Subsystem for Linux.

You will need to download the source code, digested observational data, and sample model data (Section 1.2.1). Afterwards, we describe how to install software dependencies using the conda[3] package manager (Section 1.2.2) or micromamba[4] (Section 1.2.2) and run the framework on sample model data (Section 1.2.5 and Section 1.2.6).

Throughout this document, `%` indicates the shell prompt and is followed by commands to be executed in a terminal in `fixed-width font`. Variable values are denoted by angle brackets, e.g. <*HOME*> is the path to your home directory returned by running `% echo $HOME`.

[2] https://github.com/NOAA-GFDL/MDTF-diagnostics/issues
[3] https://docs.conda.io/en/latest/
[4] https://mamba.readthedocs.io/en/latest/user_guide/micromamba.html

1.2.1 Obtaining the code

The official repo for the package's code is hosted at the NOAA-GFDL GitHub account[5]. To simplify updating the code, we recommend that all users obtain the code using git. For more in-depth instructions on how to use git, see *Git-based development workflow* (page 33).

To install the MDTF package on a local machine, open a terminal and create a directory named *mdtf*. Instructions for end-users and new developers are then as follows:

- For end users:
  1. `% cd mdtf`, then clone your fork of the MDTF repo on your machine:
     `% git clone https://github.com/<your GitHub account name>/MDTF-diagnostics`.
  2. Verify that you are on the main branch: `% git branch`.
  3. Check out the latest official release[6]:
     `% git checkout tags/v4.0.alpha`.
  4. Proceed with the installation process described below.
  5. Check out a new branch that will contain your edited config files:
     `% git checkout -b <branch name>`.
  6. Update the config files, then commit the changes:
     `% git commit -m "description of your changes"`.
  7. Push the changes on your branch to your remote fork:
     `% git push -u origin <branch name>`.
- For new POD developers:
  1. `% cd mdtf`, then clone your fork of the MDTF repo on your machine:
     `% git clone https://github.com/<your GitHub account name>/MDTF-diagnostics`.
  2. Check out the `main` branch: `% git checkout main`.
  3. Proceed with the installation process described below.
  4. Check out a new branch for your POD:
     `% git checkout -b <POD branch name>`.
  5. Edit existing files/create new files, then commit the changes:
     `% git commit -m "description of your changes"`.
  6. Push the changes on your branch to your remote fork:
     `% git push -u origin <POD branch name>`.

The path to the code directory (`.../mdtf/MDTF-diagnostics`) is referred to as <*CODE_ROOT*>. It contains the following subdirectories:

- `diagnostics/`: directory containing source code and documentation of individual PODs.
- `doc/`: source code for the documentation website.
- `shared/`: shared code and resources for use by both the framework and PODs.
- `sites/`: site-specific code and configuration files.
- `src/`: source code of the framework itself.
- `submodules/`: 3rd party software included in the framework workflow as submodules

[5] https://github.com/NOAA-GFDL/MDTF-diagnostics
[6] https://github.com/NOAA-GFDL/MDTF-diagnostics/releases/tag/v4.0.alpha

- `templates/`: runtime configuration template files
- `tests/`: general tests for the framework.
- `tools/`: helper scripts for building data catalogs and data management
- `user_scripts/`: directory for POD developers to place custom preprocessing scripts

For advanced users interested in keeping more up-to-date on project development and contributing feedback, the `main` branch of the GitHub repo contains features that haven't yet been incorporated into an official release, which are less stable or thoroughly tested.

1.2.2 Installing dependencies

Installing XQuartz on MacOS

If you're installing on a MacOS system, you will need to install XQuartz[7]. If the XQuartz executable isn't present in `/Applications/Utilities`, you will need to download and run the installer from the previous link.

The reason for this requirement is that the X11 libraries are required dependencies[8] for the NCL scripting language, even when it's run non-interactively. Because the required libraries cannot be installed through conda (next section), this installation needs to be done as a manual step.

Managing dependencies with the conda package manager

The MDTF framework code is written in Python 3.11, but supports running PODs written in a variety of scripting languages and combinations of libraries. To ensure that the correct versions of these dependencies are installed and available, we use conda[9], a free, open-source package manager. Conda is one component of the Miniconda[10] and Anaconda[11] python distributions, so having Miniconda/Anaconda is sufficient but not necessary.

For maximum portability and ease of installation, we recommend that all users manage dependencies through conda using the steps below, even if they have independent installations of the required languages. A complete installation of all dependencies will take roughly 5 Gb, less if you've already installed some of the dependencies through conda. The location of this installation can be changed with the `--conda_root` and `--env_dir` flags described below.

Users may install their own copies of Anaconda/Miniconda on their machine, or use a centrally-installed version managed by their institution. Note that installing your own copy of Anaconda/Miniconda will re-define the default locations of the conda executable and environment directory defined in your *.bash_profile*, .bashrc`, or *.cshrc* file if you have previously used a version of conda managed by your institution, so you will have to re-create any environments made using central conda installations.

[7] https://www.xquartz.org/
[8] https://www.ncl.ucar.edu/Download/macosx.shtml#InstallXQuartz
[9] https://docs.conda.io/en/latest/
[10] https://docs.conda.io/en/latest/miniconda.html
[11] https://www.anaconda.com/

Installing the conda package manager

In this section, we install the conda package manager if it's not already present on your system.

- To determine if conda is installed, run `% conda info` as the user who will be using the package. The package has been tested against versions of conda >= 4.11.0. If a pre-existing conda installation is present, continue to the following section to install the package's environments. These environments will co-exist with any existing installation.

> **Note:** **Do not** reinstall Miniconda/Anaconda if it's already installed for the user who will be running the package: the installer will break the existing installation (if it's not managed with, e.g., environment modules.)

- If `% conda info` doesn't return anything, you will need to install conda. We recommend doing so using the Miniconda installer (available here[12]) for the most recent version of python 3.
- Follow the conda installation instructions[13] appropriate to your system.
- Toward the end of the installation process, enter "yes" at "Do you wish the installer to initialize Miniconda3 by running conda init?" (or similar) prompt. This will allow the installer to add the conda path to the user's shell login script (e.g., `~/.bashrc` or `~/.cshrc`). It's necessary to modify your login script due to the way conda is implemented.
- Start a new shell to reload the updated shell login script.

Installing micromamba

Micromamaba installation instructions[14]

Installing the package's conda environments

In this section we use conda to install the versions of the language interpreters and third-party libraries required by the package's diagnostics.

- First, determine the location of your conda/micromamba installation by running `% conda info --base` or `% micromamba info` as the user who will be using the package. This path will be referred to as <*CONDA_ROOT*> or <*MICROMAMBA_ROOT*> below.
- If you don't have write access to <*CONDA_ROOT*>/<*MICROMAMBA_ROOT*> (for example, if conda has been installed for all users of a multi-user system), you will need to tell conda to install its files in a different, writable location. You can also choose to do this out of convenience, e.g. to keep all files and programs used by the MDTF package together in the `mdtf` directory for organizational purposes. This location will be referred to as <*CONDA_ENV_DIR*> below.

To display information about all of the options in the conda_env_setup.sh and micromamba_env_setup.sh environment installation scripts, run

```
% cd <CODE_ROOT>
% ./src/conda/conda_env_setup.sh [-h|--help]
% ./src/conda/micromamba_env_setup.sh [-h|--help]
```

Install all the package's conda environments with anaconda/miniconda by running

[12] https://docs.conda.io/en/latest/miniconda.html
[13] https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html
[14] https://mamba.readthedocs.io/en/latest/micromamba-installation.html#

```
% cd <CODE_ROOT>
% ./src/conda/conda_env_setup.sh --all --conda_root <CONDA_ROOT> --env_dir
↪<CONDA_ENV_DIR>
```

The names of all conda environments used by the package begin with "_MDTF", so as not to conflict with other environments in your conda installation. The installation process should finish within ten minutes.

Substitute the paths identified above for <*CONDA_ROOT*> and <*CONDA_ENV_DIR*>.

If the `--env_dir` flag is omitted, the environment files will be installed in your system's conda's default location (usually <*CONDA_ROOT*>/envs).

Install all the package's conda environments with micromamba by running

```
% cd <CODE_ROOT>
% ./src/conda/micromamba_env_setup.sh --all --micromamba_root <MICROMAMBA_ROOT>
↪ --micromamba_exe <MICROMAMBA_EXE> --env_dir <CONDA_ENV_DIR>
```

<*MICROMAMBA_ROOT*> is the path to the micromamba installation on your system (e.g., /home/${USER}/micromamba)

<*MICROMAMBA_EXE*> is the path to the micromamba executable on your system (e.g., /home/${USER}/.local/bin/micromamba)

Note: Micromamba is required to install the conda environments on machines with Apple M-series chips. NCL and R do not provide package support these systems, and only python-based environments and PODs will work. Install the base and python3_base environments individually on M-series Macs by running

```
% cd <CODE_ROOT>
% ./src/conda/micromamba_env_setup.sh -e base --micromamba_root <MICROMAMBA_ROOT> --
↪micromamba_exe <MICROMAMBA_EXE> --env_dir <CONDA_ENV_DIR>
% ./src/conda/micromamba_env_setup.sh -e python3_base --micromamba_root <MICROMAMBA_ROOT>
↪ --micromamba_exe <MICROMAMBA_EXE> --env_dir <CONDA_ENV_DIR>
```

Note: After installing the framework-specific conda environments, you shouldn't alter them manually (i.e., never run `conda update` on them). To update the environments after an update to a new release of the framework code, re-run the above commands.

These environments can be uninstalled by deleting their corresponding directories under <*CONDA_ENV_DIR*> (or <*CONDA_ROOT*>/envs/).

Location of the installed executable

The script used to install the conda environments in the previous section creates a script named `mdtf` in the MDTF-diagnostics directory. This script is the executable you'll use to run the package and its diagnostics. To test the installation, run

```
% cd <CODE_ROOT>
% ./mdtf --help
```

The output should be

```
Usage: MDTF-diagnostics [OPTIONS]

A community-developed package to run Process Oriented Diagnostics on weather
and climate data

Options:
    -v, --verbose          Enables verbose mode.
    -f, --configfile PATH  Path to the runtime configuration file  [required]
    --help                 Show this message and exit.
```

1.2.3 Creating synthetic data for example_multicase and other 4th generation and newer PODs that use ESM-intake catalogs

To generate synthetic data for functionality testing, create the Conda environment from the _env_sythetic_data.yml file in src/conda, activate the environment, and install the mdtf-test-data[15] package. Then run the driver script with the desired data convention, start year, and time span. The example below generates two CMIP datasets spanning 5 years that start in 1980 and 1985. The sample data can be used to run the example_multicase POD[16] using the configuration in the multirun_config_template file[17].

```
% mamba env create --force -q -f ./src/conda/_env_synthetic_data.yml
% conda activate _MDTF_synthetic_data
% pip install mdtf-test-data
% mkdir mdtf_test_data && cd mdtf_test_data
% mdtf_synthetic.py -c CMIP --startyear 1980 --nyears 5
% mdtf_synthetic.py -c CMIP --startyear 1985 --nyears 5
```

1.2.4 Obtaining supporting data for 3rd-generation and older single-run PODs

Supporting observational data and sample model data for second and third generation single-run PODs are available via anonymous FTP from ftp://ftp.cgd.ucar.edu/archive/mdtf. The observational data is required for the PODs' operation, while the sample model data is optional and only needed for test and demonstration purposes. The files you will need to download are:

- Digested observational data (159 Mb): MDTF_v2.1.a.obs_data.tar[18].
- NCAR-CESM-CAM sample data (12.3 Gb): model.QBOi.EXP1.AMIP.001.tar[19].
- NOAA-GFDL-CM4 sample data (4.8 Gb): model.GFDL.CM4.c96L32.am4g10r8.tar[20].

The default single-run test case uses the `QBOi.EXP1.AMIP.001` sample dataset, and the `GFDL.CM4.c96L32.am4g10r8` sample dataset is only for testing the MJO Propagation and Amplitude POD. Note that the above paths are symlinks to the most recent versions of the data, and will be reported as having a size of zero bytes in an FTP client.

Download these files and extract the contents in the following directory hierarchy under the `mdtf` directory:

```
mdtf
├── MDTF-diagnostics ( = <CODE_ROOT>)
```

(continues on next page)

[15] https://pypi.org/project/mdtf-test-data/
[16] https://github.com/NOAA-GFDL/MDTF-diagnostics/tree/main/diagnostics/example_multicase
[17] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/diagnostics/example_multicase/multirun_config_template.jsonc
[18] ftp://ftp.cgd.ucar.edu/archive/mdtf/MDTF_v2.1.a.obs_data.tar
[19] ftp://ftp.cgd.ucar.edu/archive/mdtf/model.QBOi.EXP1.AMIP.001.tar
[20] ftp://ftp.cgd.ucar.edu/archive/mdtf/model.GFDL.CM4.c96L32.am4g10r8.tar

(continued from previous page)

```
├── inputdata
│   ├── model
│   │   ├── GFDL.CM4.c96L32.am4g10r8
│   │   │   └── day
│   │   │       ├── GFDL.CM4.c96L32.am4g10r8.precip.day.nc
│   │   │       └── (... other .nc files )
│   │   └── QBOi.EXP1.AMIP.001
│   │       ├── 1hr
│   │       │   ├── QBOi.EXP1.AMIP.001.PRECT.1hr.nc
│   │       │   └── (... other .nc files )
│   │       ├── 3hr
│   │       │   └── QBOi.EXP1.AMIP.001.PRECT.3hr.nc
│   │       ├── day
│   │       │   ├── QBOi.EXP1.AMIP.001.FLUT.day.nc
│   │       │   └── (... other .nc files )
│   │       └── mon
│   │           ├── QBOi.EXP1.AMIP.001.PS.mon.nc
│   │           └── (... other .nc files )
│   └── obs_data ( = <OBS_DATA_ROOT>)
│       ├── (... supporting data for individual PODs )
```

Note that `mdtf` now contains both the `MDTF-diagnostics` and `inputdata` directories.

You can put the observational data and model output in different locations, e.g. for space reasons, by changing the paths given in `OBS_DATA_ROOT` as described below in Section 1.2.5.

1.2.5 Configuring framework paths

In order to run the diagnostics in the package, it needs to be provided with paths to the data and code dependencies installed above. In general, there are two equivalent ways to configure any setting for the package:

- All settings are configured with command-line flags. The full documentation for the command line interface is at *Command-line options* (page 133).
- Long lists of command-line options are cumbersome, and many of the settings (such as the paths to data that we set here) don't change between different runs of the package. For this purpose, any command-line setting can also be provided in an input configuration file.
- The two methods of setting options can be freely combined. Any values set explicitly on the command line will override those given in the configuration file.

For the remainder of this section, we describe how to edit and use configuration files, since the paths to data, etc., we need to set won't change.

Runtime configuration file json and yaml templates are located in the templates[21] directory. You can customize either template depending on your preferences; save a copy of the file at <*config_file_path*> and open it in a text editor. The following paths need to be configured before running the framework:

- `DATA_CATALOG`: set to the path of the ESM-intake data catalog with model input data
- `OBS_DATA_ROOT`: set to the location of input observational data if you are running PODs that require observational datasets (e.g., ../inputdata/obs_data).
- `conda_root`: should be set to the location of your conda installation: the value of <*CONDA_ROOT*> that was used in Section 1.2.2

[21] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/templates

- `conda_env_root`: set to the location of the conda environments (should be the same as <*CONDA_ENV_DIR*> in Section 1.2.2)
- `micromamba_exe`: Set to the full path to micromamba executable on your system if you are using micromamba to manage the conda environments
- `OUTPUT_DIR`: should be set to the location you want the output files to be written to (default: `mdtf/wkdir/`; will be created by the framework). The output of each run of the framework will be saved in a different subdirectory in this location.

In *Running the package on your data* (page 10), we describe more of the most important configuration options for the package, and in particular how you can configure the package to run on different data. A complete description of the configuration options is at *Command-line options* (page 133), or can be obtained by running `% ./mdtf --help`.

1.2.6 Running the package on the example_multicase POD with synthetic CMIP model data

You are now ready to run the example_multicase POD on the synthetic CMIP data. which is saved at <*config_file_path*> as described in the previous section.

```
% cd <CODE_ROOT>
% ./mdtf -f <config_file_path>
```

The first few lines of output will be

```
POD convention and data convention are both no_translation. No data translation will be␣
↪performed for case CMIP_Synthetic_r1i1p1f1_gr1_19800101-19841231.
POD convention and data convention are both no_translation. No data translation will be␣
↪performed for case CMIP_Synthetic_r1i1p1f1_gr1_19850101-19891231.
Preprocessing data for example_multicase
```

Run time may be up to 10-20 minutes, depending on your system. The final lines of output should be:

```
SubprocessRuntimeManager: completed all PODs.
Checking linked output files for <#O2Lr:example_multicase>.
No files are missing.
```

Process finished with exit code 0

The output are written to a directory named `MDTF_Output` in <*OUTPUT_DIR*>. The results are presented as a series of web pages, with the top-level page named index.html. To view the results in a web browser (e.g., Google Chrome, Firefox) run

```
% firefox <OUTPUT_DIR>/MDTF_Output/example_multicase/index.html &
```

In *Running the package on your data* (page 10), we describe further options to customize how the package is run.

1.3 Running the package on your data

In this section we describe how to proceed beyond running the simple test case described in the *previous section* (page 2), in particular how to run the framework on your own model data.

1.3.1 Preparing your data for use by the package

You have multiple options for organizing or setting up access to your model's data in a way that the framework can recognize. This task is performed by a "data source," a code plug-in that handles obtaining model data from a remote location for analysis by the PODs.

In order to identify what variable names correspond to the physical quantities requested by each POD, the LocalFile data source requires that model data follow one of several recognized variable naming conventions defined by the package. The currently recognized conventions are:

- `CMIP`: Variable names and units as used in the CMIP6[22] data request[23]. There is a web interface[24] to the request. Data from any model that has been published as part of CMIP6 (e.g., made available via ESGF[25]) should follow this convention.
- `CESM`: Variable names and units used in the default output of models developed at the National Center for Atmospheric Research[26], such as CAM[27] (all versions) and CESM2[28].
- `GFDL`: Variable names and units used in the default output of models developed at the Geophysical Fluid Dynamics Laboratory[29], such as AM4[30], CM4[31] and SPEAR[32].

The names and units for the variables in the model data you're adding need to conform to one of the above conventions in order to be recognized by the LocalFile data source. For models that aren't currently supported, the workaround we recommend is to generate `CMIP`-compliant data by postprocessing model output with the CMOR[33] tool. We hope to offer support for the naming conventions of a wider range of models in the future.

Generating an ESM-intake catalog of your model dataset

The MDTF-diagnostics uses intake-ESM[34] catalogs and APIs to access model datasets and verify POD data requirements. The MDTF-diagnostics package provides a basic catalog_builder script[35] that uses ecgtools[36] APIs to generate data catalogs. The NOAA-GFDL workflow team also maintains an intake-ESM catalog builder[37] that uses the directory structure to generate data catalogs. It is optimized for the files stored on GFDL systems, but can be configured to generate catalogs on a local file system.

[22] https://www.wcrp-climate.org/wgcm-cmip/wgcm-cmip6
[23] https://doi.org/10.5194/gmd-2019-219
[24] http://clipc-services.ceda.ac.uk/dreq/index.html
[25] https://esgf-node.llnl.gov/projects/cmip6/
[26] https://ncar.ucar.edu
[27] https://www.cesm.ucar.edu/models/cesm2/atmosphere/
[28] https://www.cesm.ucar.edu/models/cesm2/
[29] https://www.gfdl.noaa.gov/
[30] https://www.gfdl.noaa.gov/am4/
[31] https://www.gfdl.noaa.gov/coupled-physical-model-cm4/
[32] https://www.gfdl.noaa.gov/spear/
[33] https://cmor.llnl.gov/
[34] https://intake-esm.readthedocs.io/en/stable/
[35] https://github.com/NOAA-GFDL/MDTF-diagnostics/tree/main/tools/catalog_builder
[36] https://ecgtools.readthedocs.io/en/latest/
[37] https://github.com/aradhakrishnanGFDL/CatalogBuilder

Adding your observational data files

If you have observational data you want to analyze available on a locally mounted disk, we recommend creating symlinks[38] that have the needed filenames, rather than making copies of the data files. For example,

```
% mkdir -p inputdata/obs_data/[pod name]
% ln -s <path> inputdata/obs_data/[pod name]/[file name]
```

will create a symbolic link to the file at <*path*> that follows the filename convention used by this data source:

```
inputdata
├── obs_data ( = <OBS_DATA_ROOT>)
│   ├── example
│       ├── example file
```

1.3.2 Running the package on your data

How to configure the package

All configuration options for the package options are set in a JSON or Yaml configuration file passed to the package with the `-f` flag. An example of this input file is given in templates/runtime_config.jsonc[39], which you used *previously* (page 9) to run the package on test data. We recommend using one of the files as a template, copying it, and customizing it as needed.

Options controlling the analysis

The configuration options required to specify what analysis the package should do are:

- **pod_list**: (list of strings) comma-separated list of PODs to run with the framework
- **case_list**: Main block with the information for each model data case to run with the framework The block for each case is a string with the name of each model simulation (case). Note that there is no explicit *CASENAME* paramater in the configuration file; the framework will define each *CASENAME* key using string value that defines the case block.
  - **model**: (string) name of the model for each case
  - **Convention**: (string) convention of case; ["CMIP" | "CESM" | "GFDL"]
  - **startdate**: (string with format <*YYYYMMDD*> or <*YYYYMMDDHHmmss>) The starting date of the analysis period
  - **enddate** (string with format <*YYYYMMDD*> or <*YYYYMMDDHHmmss>) The end date of the analysis period.

An error will be raised if the data provided for any requested variable doesn't span the date range defined by **startdate** and **enddate**

[38] https://en.wikipedia.org/wiki/Symbolic_link
[39] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/templates/runtime_config.jsonc

Options for data management

- **DATA_CATALOG**: (string; *required*) Full or relative path to the model data ESM-intake catalog .json header file
- **OBS_DATA_ROOT**: (string; optional) Full or relative path to Parent directory containing observational data. Must be set if running PODs that have required observational datasets.
- **WORK_DIR**: (string; required) Full or relative path to working directory
- **OUTPUT_DIR**: (string; optional) Full or relative path to output directory; The results of each run of the framework will be put in an *MDTF_output* subdirectory of this directory. Defaults to **WORK_DIR** if blank.
- **conda_root**: (string; required) Location of the Anaconda/miniconda or micromamba installation to use for managing package dependencies (path returned by running *conda info –base* or *micromamba info*.)
- **conda_env_root**: (string; required) Directory containing the framework-specific conda environments. This should be equal to the "–env_dir" flag passed to *conda_env_setup.sh*
- **micromambe_exe** (string; required if using micromamba to manage conda environments) Full path to the micromamba executable

Options for workflow control

- **run_pp**: (boolean) Set to *true* to run the preprocessor; default *true*
- **translate_data**: (boolean) Set to *true* to perform data translation; default *true*
- **save_ps**: (boolean) Set to *true* to have PODs save postscript figures in addition to bitmaps; default *false*
- **large_file**: (boolean) Set to *true* for files > 4 GB. The framework will write processed netCDF files in *NETCDF4_CLASSIC* format; if *false* files are written in *NETCDF4* format; default *false*
- **save_pp_data**: (boolean) set to *true* to retain processed data in the *OUTPUT_DIR* after preprocessing. If *false*, delete processed data after POD output is finalized; default *true*
- **make_variab_tar**: (boolean) Set to *true* to save HTML and bitmap plots in a .tar file; default *false*
- **make_multicase_figure_html**: (boolean) Set to *true* to auto-generate html output for multiple figures per case; default *false*
- **overwrite**: (boolean) Set to *true* to overwrite newest existing *OUTPUT_DIR* from a previous run; default *false*
- **user_pp_scripts**: (list of strings) comma-separated Python list of strings with custom preprocessing scripts to include in the workflow. Add any custom script(s) you want to run to the user_scripts[40] directory of your copy of the MDTF-diagnostics repository. The scripts will run even if the list is populated whether **run_pp** is set to *true* or *false*.

Running the package

From this point, the instructions for running the package are the same as for *running it on the sample data* (page 9), assuming you've set the configuration options by editing a copy of the configuration file template at `templates/runtime_config.jsonc

<https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/templates/runtime_config.jsonc>`__. The package is run in the

same way:

[40] https://github.com/NOAA-GFDL/MDTF-diagnostics/tree/main/user_scripts

```
% cd <CODE_ROOT>
% ./mdtf -f <new config file path>
```

The output of the package will be saved as a series of web pages in a directory named MDTF_output/[pod_name] in <*OUTPUT_DIR*>.

If you run the package multiple times with the same configuration values and **overwrite** set to *false, the suffixes ".v1", ".v2", etc. will be added to duplicate *MDTF_output* directory names.

1.4 Running the MDTF-diagnostics package in "multirun" mode

Version 3 and later of the MDTF-diagnostics package provides support for "multirun" diagnostics that analyze output from multiple model and/or observational datasets. At this time, the multirun implementation is experimental, and may only be run on appropriately-formatted PODs. "Single-run" PODs that analyze one model dataset and/or one observational dataset must be run separately because the configuration for single-run and multi-run analyses is different. Users and developers should open issues when they encounter bugs or require additional features to support their PODs, or run existing PODs on new datasets.

1.4.1 The example_multicase POD and configuration

A multirun test POD called *example_multicase* is available in diagnostics/example_multicase that demonstrates how to configure "multirun" diagnostics that analyze output from multiple datasets. The multirun_config_template.jsonc file[41] contains separate `pod_list` and `case_list` blocks. As with the single-run configuration, the `pod_list` may contain multiple PODs separated by commas. The `case_list` contains multiple blocks of information for each case that the POD(s) in the `pod_list` will analyze. The `CASENAME`, `convention`, `startdate`, and `enddate` attributes must bedefined for each case. The `convention` must be the same for each case, but `startdate` and `enddate` may differ among cases. Directions for generating the synthetic data in the configuration file are provided in the file comments, and in the quickstart section of the README file[42]

1.4.2 POD output

The framework defines a root directory `$WORK_DIR/[POD name]` for each POD in the pod_list. `$WORK_DIR/[POD name]` contains the the main framework log files, and subdirectories for each case. Temporary copies of processed data for each case are placed in `$WORK_DIR/[CASENAME]/[data output frequency]`. The pod html file is written to `$OUTPUT_DIR/[POD name]/[POD_name].html` (`$OUTPUT_DIR` defaults to `$WORK_DIR` if it is not defined), and the output figures are placed in `$OUTPUT_DIR/[POD name]/model` depending on how the paths are defined in the POD's html template.

Note that an obs directory is created by default, but will be empty unless the POD developer opts to use an observational dataset and write observational data figures to this directory. Figures that are generated as .eps files before conversion to .png files are written to `$WORK_DIR/[POD name]/model/PS`.

[41] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/diagnostics/example_multicase/multirun_config_template.jsonc
[42] https://github.com/NOAA-GFDL/MDTF-diagnostics#5-run-the-framework-in-multi_run-mode-under-development

1.4.3 Multirun environment variables

Multirun PODs obtain information for environment variables for the case and variable attributes described in the *configuration section* (page 10) from a yaml file named *case_info.yaml* that the framework generates at runtime. The *case_info.yaml* file is written to `$WOR_DIR/[POD name]`, and has a corresponding environment variable *case_env_file* that the POD uses to parse the file. The *example_multicase.py* script demonstrates to how to read the environment variables from *case_info.yaml* using the *case_env_file* environment variable into a dictionary, then loop through the dictionary to obtain the post-processed data for analysis. An example *case_info.yaml* file with environment variables defined for the synthetic test data is located in the *example_multicase* directory.

CHAPTER

TWO

DEVELOPER INFORMATION

2.1 POD requirements

This section lists all the steps that need to be taken in order to submit a POD for inclusion in the MDTF framework.

2.1.1 Code and documentation submission

The material in this section must be submitted though a pull request[43] to the NOAA-GFDL GitHub repo[44]. This is described in *Git-based development workflow* (page 33).

Use the example_multicase POD[45]
as a reference for how each component of the submission should be structured.

The POD feature must be up-to-date with the NOAA-GFDL main branch, with no outstanding merge conflicts. See *Git-based development workflow* (page 33) for instructions on syncing your fork with NOAA-GFDL, and pulling updates from the NOAA-GFDL main branch into your feature branch.

POD source code

All scripts should be placed in a subdirectory of `diagnostics/`. Among the scripts, there should be 1) a main driver script, 2) a template html, and 3) a `settings.jsonc` file. The POD directory and html template should be named after your POD's short name.

For instance, `diagnostics/convective_transition_diag/` contains its driver script `convective_transition_diag.py`, `convective_transition_diag.html`, and `settings.jsonc`, etc.

The framework will call the driver script, which calls the other scripts in the same POD directory.

If you need a new Conda environment, add a new .yml file to `src/conda/`, and install the environment using the `conda_env_setup.sh` or `micromamba_env_setup.sh` scripts as described in the *Getting Started* (page 2).

[43] https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests
[44] https://github.com/NOAA-GFDL/MDTF-diagnostics
[45] https://github.com/NOAA-GFDL/MDTF-diagnostics/tree/main/diagnostics/example_multicase

POD settings file

The format of this file is described in *POD settings file summary* (page 24).

POD html template for output

The html template will be copied by the framework into the output directory to display the figures generated by the POD. You should be able to create a new html template by simply copying and modifying the example templates from existing PODs even without prior knowledge about html syntax. If you have a

POD documentation

The documentation for the framework is automatically generated using sphinx[46], which works with files in reStructured text[47] (reST, `.rst`) format.

Use the example_multicase POD documentation[48] as a template for the information required for your POD, by modifying its .rst source code[49]. The documentation should include the following information:

- a one-paragraph synopsis of the POD
- the developers' contact information
- required programming language and libraries
- a brief summary of the presented diagnostics
- references in which more in-depth discussions can be found.

The .rst files and all linked figures should be placed in a `doc` subdirectory under your POD directory (e.g., `diagnostics/example_multicase/doc/`) and put the .rst file and figures inside.

The most convenient way to write and debug reST documentation is with an online editor. We recommend https://livesphinx.herokuapp.com/ because it recognizes sphinx-specific commands as well.

For reference, see the reStructured text introduction[50], quick reference[51] and in-depth guide[52].

Also see a reST syntax comparison[53] to other text formats you may be familiar with.

- For maintainability, all scripts should be self-documenting by including in-line comments.

The main driver script (e.g., `example_multicase.py`) should contain a comprehensive header providing information that contains the same items as in the POD documentation, except for the "More about this diagnostic" section.

- The one-paragraph POD synopsis (in the POD documentation) as well as a link to the full documentation should be

placed at the top of the html template (e.g., `example_multicase.html`).

[46] https://www.sphinx-doc.org/en/master/index.html
[47] https://docutils.sourceforge.io/rst.html
[48] https://mdtf-diagnostics.readthedocs.io/en/latest/sphinx_pods/example_multicase.html
[49] https://raw.githubusercontent.com/NOAA-GFDL/MDTF-diagnostics/main/diagnostics/example/doc/example.rst
[50] http://docutils.sourceforge.net/docs/user/rst/quickstart.html
[51] http://docutils.sourceforge.net/docs/user/rst/quickref.html
[52] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html
[53] http://hyperpolyglot.org/lightweight-markup

2.1.2 Sample and supporting data submission

Data hosting for the MDTF framework is currently managed manually. The data is hosted via anonymous FTP on UCAR's servers.

Digested observational or supporting data

Create a directory under `inputdata/obs_data/` named after the short name of your POD, and put all your *digested* observation data in (or more generally, any quantities that are independent of the model being analyzed).

- Requirements - Digested data should be in the form of numerical data, not figures. - The data files should be small (preferably a few MB) and just enough for producing figures for model comparison.

If you really cannot reduce the data size and your POD requires more than 1GB of space, consult with the lead team.

  - Include in the directory a "README.txt" description file with original source info.
  - Include in the directory any necessary licensing information, files, etc. (if applicable)

- Create a tar file of your obs_data directory: - Use the –hard_dereference flag so that all users can read your file. - Naming convention: $pod_name.yyyymmdd.tar, where yyyymmdd is the file creation date.

Alternatively, you may use some other version tag to allow the framework to check compatibiity between the POD code and data provided.

- Create the tar file from the inputdata directory so the file paths start with obs_data.
- Example (c-shell):

```
set pod_name = MJO_suite
set tartail  =  `date +'%Y%m%d'`
cd inputdata/obs_data
tar cfh $pod_name.$tartail.tar --hard-dereference $pod_name
```

- To check:

```
% tar tf $pod_name.$tartail.tar
MJO_suite/
MJO_suite/ERA.v200.EOF.summer-0.png
MJO_suite/ERA.u200.EOF.summer-1.png
```

After following the above instructions, please refer to the GitHub Discussion on transfering obs_data[54] or email Dani Coleman at bundy at ucar dot edu or contact your liason on the MDTF Leads Team.

Files will be posted for Guest/anonymous access : ftp://ftp.cgd.ucar.edu/archive/mdtf/obs_data_latest/{$pod_name}.latest.tar with 'latest' pointing to the date-or-version-tagged tar file

Note that prior to version 3, obs_data from all PODs was consolidated in one tar file. To assist in usability as the number of PODs grow, they will now be available individually, with the responsiblity for creating the tar files on the developer.

[54] https://github.com/NOAA-GFDL/MDTF-diagnostics/discussions/125

Sample model data

For PODs dealing with atmospheric phenomena, we recommend that you use sample data from the following sources, if applicable:

- A timeslice run of NCAR CAM5[55]
- A timeslice run of GFDL AM4[56] (contact the leads for password).

2.2 Developer installation instructions

To download and install the framework for development, follow the instructions for end users given in *Installation instructions* (page 2), with the following developer-specific modifications:

2.2.1 Obtaining the source code

POD developers should create their branches from the main branch[57] of the framework code

```
git checkout -b [POD branch name] main
```

This is the "beta test" version, used for testing changes before releasing them to end users

Developers may download the code from GitHub as described in *Obtaining the code* (page 3), but we strongly recommend that you clone the repo in order to keep up with changes in the main branch, and to simplify submitting pull requests with your POD's code. Instructions for how to do this are given in *Git-based development workflow* (page 33).

2.2.2 Installing dependencies with Conda

Regardless of development language, we strongly recommend that developers use conda to manage their language and library versions. Note that Conda is not Python-specific, but allows coexisting versioned environments of most scripting languages, including, R[58], NCL[59], Ruby[60], etc…

Python-based PODs should be written in Python 3.11 or newer. We provide a developer version of the python3_base environment (described below) that includes Jupyter and other developer-specific tools. This is not installed by default, and must be requested by passing the `--all` flag to the conda setup script:

If you are using Anaconda or miniconda to manage the conda environments, run:

```
% cd $CODE_ROOT
% ./src/conda/conda_env_setup.sh --all --conda_root $CONDA_ROOT --env_dir $CONDA_ENV_DIR
```

[55] https://www.earthsystemgrid.org/dataset/ucar.cgd.ccsm4.NOAA-MDTF.html
[56] http://data1.gfdl.noaa.gov/MDTF/
[57] https://github.com/NOAA-GFDL/MDTF-diagnostics/tree/main
[58] https://anaconda.org/conda-forge/r-base
[59] https://anaconda.org/conda-forge/ncl
[60] https://anaconda.org/conda-forge/ruby

2.2.3 Installing dependencies with Micromamba

Micromamba is a lightweight version of Anaconda. It is required to install the base and python3_base conda environments on macOS machines with Apple M-series chips. Installation instructions are available in the Micromamba Documentation[61], Once Micromamba is installed on your system, run the following to install all conda environments if you are NOT using an Apple M-series machine, where *$MICROMAMBA_ROOT* is the location of the micromamba installation, and *MICROMAMBA_EXE* is the path to the micromamba executable on your system:

```
% cd $CODE_ROOT
% ./src/conda/micromamba_env_setup.sh --all --conda_root $MICROMAMBA_ROOT --micromamba_
↪exe $MICROMAMBA_EXE --env_dir $CONDA_ENV_DIR
```

If you are using an Apple M-series machine, you can install just the base and python3_base environments:

POD development using existing Conda environments

To prevent the proliferation of dependencies, we suggest that new POD development use existing Conda environments whenever possible, e.g., python3_base[62], NCL_base[63], and R_base[64] for Python, NCL, and R, respectively.

In case you need any exotic third-party libraries, e.g., a storm tracker, consult with the lead team and create your own Conda environment following *instructions* (page 20) below.

2.2.4 Python

The framework provides the _MDTF_python3_base[65] Conda environment (recall the `_MDTF` prefix for framework-specific environment) as the generic Python environment, which you can install following the *instructions* (page 4). You can then activate this environment by running in a terminal:

```
% source activate $CONDA_ENV_DIR/_MDTF_python3_base
```

where `$CONDA_ENV_DIR` is the path you used to install the Conda environments. After you've finished working under this environment, run `% conda deactivate` or simply close the terminal.

2.2.5 Other languages

The framework also provides the _MDTF_NCL_base[66] and _MDTF_R_base[67] Conda environments as the generic NCL and R environments.

[61] https://mamba.readthedocs.io/en/latest/micromamba-installation.html
[62] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/src/conda/env_python3_base.yml
[63] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/src/conda/env_NCL_base.yml
[64] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/src/conda/env_R_base.yml
[65] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/src/conda/env_pythone3_base.yml
[66] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/src/conda/env_NCL_base.yml
[67] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/src/conda/env_R_base.yml

POD development using a new Conda environment

If your POD requires languages that aren't available in an existing environment or third-party libraries unavailable through the common conda-forge[68] and anaconda[69] channels, we ask that you notify the framework developers (since this situation may be relevant to other developers) and submit a YAML (.yml) file[70] that creates the environment needed for your POD.

- The new YAML file should be added to `src/conda/`, where you can find templates for existing environments from which you can create your own.
- The YAML filename should be `env_$your_POD_short_name.yml`.
- The first entry of the YAML file, name of the environment, should be `_MDTF_$your_POD_short_name`.
- We recommend listing conda-forge as the first channel to search, as it's entirely open source and has the largest range of packages. Note that combining packages from different channels (in particular, conda-forge and Anaconda channels) may create incompatibilities.
- We recommend constructing the list of packages manually, by simply searching your POD's code for `import` statements referencing third-party libraries. Please do *not* exporting your development environment with `% conda env export`, which gives platform-specific version information and will not be fully portable in all cases; it also does so for every package in the environment, not just the "top-level" ones you directly requested.
- We recommend specifying versions as little as possible, out of consideration for end-users: if each POD specifies exact versions of all its dependencies, conda will need to install multiple versions of the same libraries. In general, specifying a version should only be needed in cases where backward compatibility was broken or a bug affecting your POD was fixed (e.g., postscript font rendering on Mac OS with older NCL). Conda installs the latest version of each package that's consistent with all other dependencies.

2.2.6 Framework interaction with Conda environments

As described in *Running the package on the example_multicase POD with synthetic CMIP model data* (page 9), when you run the `mdtf` executable, among other things, it reads `pod_list` in `runtime_config.[jsonc | yml]` and executes POD codes accordingly. For a POD included in the list (referred to as $POD_NAME):

- The framework checks for required packages in the POD's `settings.jsonc` file in `$CODE_ROOT/diagnostics/$POD_NAME/`. The `runtime_requirements` section in `settings.jsonc` specifies the programming language(s) adopted by the POD:

  a). If purely Python 3, the framework will look for `src/conda/env_python3_base.yml` and check its content to determine whether the POD's requirements are met, and then switch to `_MDTF_python3_base` and run the POD.

  b). Similarly, if NCL or R is used, then `NCL_base` or `R_base` environment will be activated at runtime.

Note that for the 6 existing PODs depending on NCL (EOF_500hPa, MJO_prop_amp, MJO_suite, MJO_teleconnection, precip_diurnal_cycle, and Wheeler_Kiladis), Python is also used but merely as a wrapper. Thus the framework will switch to `_MDTF_NCL_base` when seeing both NCL and Python in `settings.jsonc`.

The framework verifies PODs' requirements via looking for the YAML files and their contents. Thus if you choose to selectively install conda environments using the `--env` flag (*Installing dependencies* (page 4)), remember to install all the environments needed for the PODs you're interested in, and that `_MDTF_base` is mandatory for the framework's operation.

[68] https://conda-forge.org/feedstocks/
[69] https://docs.anaconda.com/anaconda/packages/pkg-docs/
[70] https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#creating-an-environment-file-manually

For instance, the minimal installation for running the `EOF_500hPa` and `convective_transition_diag PODs` requres `_MDTF_base` (mandatory), `_MDTF_NCL_base` (because of b), and `_MDTF_convective_transition_diag` (because of 1). These can be installed by passing `base`, `NCL_base`, and `convective_transition_diag` to the `--env` flag one at a time (*Installing dependencies* (page 4)).

2.2.7 Testing with a new Conda environment

If you've updated an existing environment or created a new environment (with corresponding changes to the YAML file), verify that your POD works.

Recall how the framework finds a proper Conda environment for a POD. First, it searches for an environment matching the POD's short name. If this fails, it then looks into the POD's `settings.jsonc` and prepares a generic environment depending on the language(s). Therefore, no additional steps are needed to specify the environment if your new YAML file follows the naming conventions above (in case of a new environment) or your `settings.jsonc` correctly lists the language(s) (in case of updating an existing environment).

- For an updated environment, first, uninstall it by deleting the corresponding directory under `$CONDA_ENV_DIR`.
- Re-install the environment using the `conda_env_setup.sh` script as described in the *installation instructions* (page 4), or create the new environment for you POD:

```
% cd $CODE_ROOT
% ./src/conda/conda_env_setup.sh --env $your_POD_short_name --conda_root
↪$CONDA_ROOT --env_dir $CONDA_ENV_DIR
```

  Or, if using micromamba:

```
% cd $CODE_ROOT
% ./src/conda/conda_env_setup.sh --env $your_POD_short_name --micromamba_
↪root $MICROMAMBA_ROOT --env_dir $CONDA_ENV_DIR
```

Have the framework run your POD on suitable test data.

1. Add your POD's short name to the `pod_list` section of the configuration input file (template: `templates/runtime_config.[jsonc | yml]`).
2. Prepare the test data as described in *Running the package on your data* (page 10).

2.3 POD development guidelines

2.3.1 Admissible languages

The framework itself is written in Python, and can call PODs written in any scripting language. However, Python support by the lead team will be "first among equals" in terms of priority for allocating developer resources, etc.

- To achieve portability, the MDTF **cannot** accept PODs written in closed-source languages (e.g., MATLAB and IDL; try Octave[71] and GDL[72] if possible). We also **cannot** accept PODs written in compiled languages (e.g., C or Fortran): installation would rapidly become impractical if users had to check compilation options for each POD.
- Python is strongly encouraged for new PODs; PODs funded through the CPO grant are requested to be developed in Python. Python version >= 3.11 is required.

[71] https://www.gnu.org/software/octave/
[72] https://github.com/gnudatalanguage/gdl

- If your POD was previously developed in NCL or R (and development is *not* funded through a CPO grant), you do not need to re-write existing scripts in Python 3 if doing so is likely to introduce new bugs into stable code, especially if you're unfamiliar with Python.
- If scripts were written in closed-source languages, translation to Python 3.11 or above is required.

2.3.2 Preparation for POD implementation

We assume that, at this point, you have a set of scripts, written in languages consistent with the framework's open source policy, that a) read in model data, b) perform analysis, and c) output figures. Here are 3 steps to prepare your scripts for POD implementation.

We recommend running the framework on the sample model data again with both `save_ps` and `save_nc` in the configuration input `src/default_tests.jsonc` set to `true`. This will preserve directories and files created by individual PODs in the output directory, which could come in handy when you go through the instructions below, and help understand how a POD is expected to write output.

- Give your POD an official name (e.g., *Convective Transition*; referred to as `long_name`) and a short name (e.g., *convective_transition_diag*). The latter will be used consistently to name the directories and files associated with your POD, so it should (1) loosely resemble the long_name, (2) avoid space bar and special characters (!@#$%^&*), and (3) not repeat existing PODs' name (i.e., the directory names under `diagnostics/`). Try to make your POD's name specific enough that it will be distinct from PODs contributed now or in the future by other groups working on similar phenomena.
- If you have multiple scripts, organize them so that there is a main driver script calling the other scripts, i.e., a user only needs to execute the driver script to perform all read-in data, analysis, and plotting tasks. This driver script should be named after the POD's short name (e.g., `convective_transition_diag.py`).
- You should have no problem getting scripts working as long as you have (1) the location and filenames of model data, (2) the model variable naming convention, and (3) where to output files/figures. The framework will provide these as *environment variables* that you can access (e.g., using `os.environ` in Python, or `getenv` in NCL). *DO NOT* hard code these paths/filenames/variable naming convention, etc., into your scripts. See the complete list of environment variables supplied by the framework.
- Your scripts should not access the internet or other networked resources.

2.3.3 An example of using framework-provided environment variables

The framework provides a collection of environment variables, mostly in the format of strings but also some numbers, so that you can and *MUST* use in your code to make your POD portable and reusable.

For instance, using 3 of the environment variables provided by the framework, `CASENAME`, `DATADIR`, and `pr_var`, the full path to the hourly precipitation file can be expressed as

```
MODEL_OUTPUT_DIR = os.environ["DATADIR"]+"/1hr/"
pr_filename = os.environ["CASENAME"]+"."+os.environ["pr_var"]+".1hr.nc"
pr_filepath = MODEL_OUTPUT_DIR + pr_filename
```

You can then use `pr_filepath` in your code to load the precipitation data.

Note that in Linux shell or NCL, the values of environment variables are accessed via a `$` sign, e.g., `os.environ["CASENAME"]` in Python is equivalent to `$CASENAME` in Linux shell/NCL.

2.3.4 Relevant environment variables

The environment variables most relevant for a POD's operation are:

- `POD_HOME`: Path to directory containing POD's scripts, e.g., `diagnostics/convective_transition_diag/`.
- `OBS_DATA`: Path to directory containing POD's supporting/digested observation data, e.g., `inputdata/obs_data/convective_transition_diag/`.
- `DATADIR` (deprecated; PODs written for MDTF-diagnostics v3.5 and earlier): Path to directory containing model data files for one case/experiment, e.g., `inputdata/model/QBOi.EXP1.AMIP.001/`.
- **`WORK_DIR`: Path to directory for POD to output files. Note that **this is the only directory a POD is allowed** to write its output**. e.g., `wkdir/MDTF_QBOi.EXP1.AMIP.001_1977_1981/convective_transition_diag/`.

  1. Output figures to `$WORK_DIR/obs/` and `$WORK_DIR/model/` respectively.

  2. `$WORKK_DIR/obs/PS/` and `$WORK_DIR/model/PS/`: If a POD chooses to save vector-format figures, save them as `EPS` under these two directories. Files in these locations will be converted by the framework to `PNG` for HTML output. Caution: avoid using `PS` because of potential bugs in recent `matplotlib` and converting to PNG.

  3. `$WORK_DIR/obs/netCDF/` and `$WORK_DIR/model/netCDF/`: If a POD chooses to save digested data for later analysis/plotting, save them in these two directories in `NetCDF`.

Note that (1) values of `POD_HOME`, `OBS_DATA`, and `WK_DIR` change when the framework executes different PODs; (2) the `WK_DIR` directory and subdirectories therein are automatically created by the framework. **Each POD should output files as described here** so that the framework knows where to find what, and also for the ease of code maintenance.

More environment variables for specifying model variable naming convention can be found in the `data/fieldlist_$convention.jsonc` files. Also see the list of environment variables supplied by the framework.

2.3.5 Guidelines for testing your POD

Test before distribution. Find people (eg, nearby postdocs/grads and members from other POD-developing groups) who are not involved in your POD's implementation and are willing to help. Give the tar files and point your GitHub repo to them. Ask them to try running the framework with your POD following the Getting Started instructions. Ask for comments on whether they can understand the documentation.

Test how the POD fails. Does it stop with clear errors if it doesn't find the files it needs? How about if the dates requested are not presented in the model data? Can developers run it on data from another model? Here are some simple tests you should try:

- If your POD uses observational data, move the `inputdata` directory around. Your POD should still work by simply updating the values of `OBS_DATA_ROOT` in the runtime configuration file.
- Try to run your POD with a different set of model data.
- If you have problems getting another set of data, try changing the files' `CASENAME` and variable naming convention. The POD should work by updating `CASENAME` and `convention` in the configuration input.
- Try your POD on a different machine. Check that your POD can work with reasonable machine configuration and computation power, e.g., can run on a machine with 32 GB memory, and can finish computation in 10 min. Will memory and run time become a problem if one tries your POD on model output of high spatial resolution and temporal frequency (e.g., avoid memory problem by reading in data in segments)? Does it depend on a particular version of a certain library? Consult the lead team if there's any unsolvable problems.

2.3.6 Other tips on implementation

1. Structure of the code package: Implementing the constituent PODs in accordance with the structure described in earlier sections makes it easy to pass the package (or just part of it) to other groups.
2. Robustness to model file/variable names: Each POD should be robust to modest changes in the file/variable names of the model output; see *Getting Started* (page 10) regarding the model data filename structure, *An example of using framework-provided environment variables* (page 22) regarding using the environment variables and robustness tests. Also, it would be easier to apply the code package to a broader range of model output.
3. Save digested data after analysis: Can be used, e.g., to save time when there is a substantial computation that can be re-used when re-running or re-plotting diagnostics.
4. Self-documenting: For maintenance and adaptation, to provide references on the scientific underpinnings, and for the code package to work out of the box without support.
5. Handle large model data: The spatial resolution and temporal frequency of climate model output have increased in recent years. As such, developers should take into account the size of model data compared with the available memory. For instance, the example POD precip_diurnal_cycle and Wheeler_Kiladis only analyze part of the available model output for a period specified by the environment variables `startdate` and `enddate`, and the convective_transition_diag module reads in data in segments.
6. Basic vs. advanced diagnostics (within a POD): Separate parts of diagnostics, e.g., those might need adjustment when model performance out of obs range.
7. Avoid special characters (`!@#$%^&*`) in file/script names.

See *Running the package on the example_multicase POD with synthetic CMIP model data* (page 9) and :doc:` framework operation walkthrough <dev_walkthrough>` for details on how the package is called. See the *command line reference* (page 133) for documentation on command line options (or run `mdtf --help`).

Avoid making assumptions about the machine on which the framework will run beyond what's listed here; a development priority is to interface the framework with cluster and cloud job schedulers to enable individual PODs to run in a concurrent, distributed manner.

2.4 POD settings file summary

This page gives a quick introduction to how to write the settings file for your POD. See the full documentation on this file format for a complete list of all the options you can specify.

2.4.1 Overview

The MDTF framework can be viewed as a "wrapper" for your code that handles data fetching and munging. Your code communicates with this wrapper in two ways:

- The *settings file* is where your code talks to the framework: when you write your code, you document what model data

your code uses and what format it expects it in. When the framework is run, it will fulfill the requests you make here (or tell the user what went wrong). - When your code is run, the framework talks to it by setting *environment variables* (page 139)

containing paths to the data files and other information specific to the run.

In the settings file, you specify what model data your diagnostic uses in a vocabulary you're already familiar with:

- The CF conventions[73] for standardized variable names and units.
- The netCDF4 (classic) data model, in particular the notions of variables[74] and dimensions[75] as they're used in a netCDF file.

2.4.2 Example

```
// Any text to the right of a '//' is a comment
{
  "settings" : {
    "long_name": "My example diagnostic",
    "driver": "example_diagnostic.py",
    "realm": "atmos",
    "runtime_requirements": {
      "python": ["numpy", "matplotlib", "netCDF4"]
    }
  },
  "data" : {
    "frequency": "day"
  },
  "dimensions": {
    "lat": {
      "standard_name": "latitude"
    },
    "lon": {
      "standard_name": "longitude"
    },
    "plev": {
      "standard_name": "air_pressure",
      "units": "hPa",
      "positive": "down"
    },
    "time": {
      "standard_name": "time",
      "units": "day"
    }
  },
  "varlist" : {
    "my_precip_data": {
      "standard_name": "precipitation_flux",
      "units": "kg m-2 s-1",
      "dimensions" : ["time", "lat", "lon"]
    },
    "my_3d_u_data": {
      "standard_name": "eastward_wind",
      "units": "m s-1",
      "dimensions" : ["time", "plev", "lat", "lon"]
    }
  }
}
```

[73] http://cfconventions.org/
[74] https://www.unidata.ucar.edu/software/netcdf/workshops/2010/datamodels/NcVars.html
[75] https://www.unidata.ucar.edu/software/netcdf/workshops/2010/datamodels/NcDims.html

2.4.3 Settings section

This is where you describe your diagnostic and list the programs it needs to run.

long_name:
Display name of your diagnostic, used to describe your diagnostic on the top-level index.html page. Can contain spaces.

driver:
Filename of the driver script the framework should call to run your diagnostic.

realm:
One or more of the eight CMIP6 modeling realms (aerosol, atmos, atmosChem, land, landIce, ocean, ocnBgchem, seaIce) describing what data your diagnostic uses. This is give the user an easy way to, eg, run only ocean diagnostics on data from an ocean model. Realm can be specified in the *settings* ` section, or specified separately for each variable in the *varlist* section.

runtime_requirements:
This is a list of key-value pairs describing the programs your diagnostic needs to run, and any third-party libraries used by those programs.

- The *key* is program's name, eg. languages such as "python[76]" or "ncl[77]" etc. but also any utilities such as "ncks[78]", "cdo[79]", etc.
- The *value* for each program is a list of third-party libraries in that language that your diagnostic needs. You do *not* need to list built-in libraries: eg, in python, you should to list numpy[80] but not math[81]. If no third-party libraries are needed, the value should be an empty list.

pod_env_vars:
object[82], optional. Names and values of shell environment variables used by your diagnostic, *in addition* to those supplied by the framework. The user can't change these at runtime, but this can be used to set site-specific installation settings for your diagnostic (eg, switching between low- and high-resolution observational data depending on what the user has chosen to download). Note that environment variable values must be provided as strings.

2.4.4 Data section

This section contains settings that apply to all the data your diagnostic uses. Most of them are optional.

frequency:
A string specifying a time span, used e.g. to describe how frequently data is sampled. It consists of an optional integer (if omitted, the integer is assumed to be 1) and a units string which is one of `hr`, `day`, `mon`, `yr` or `fx`. `fx` is used where appropriate to denote time-independent data. Common synonyms for these units are also recognized (e.g. `monthly`, `month`, `months`, `mo` for `mon`, `static` for `fx`, etc.)

[76] https://www.python.org/
[77] https://www.ncl.ucar.edu/
[78] http://nco.sourceforge.net/
[79] https://code.mpimet.mpg.de/projects/cdo
[80] https://numpy.org/
[81] https://docs.python.org/3/library/math.html
[82] https://docs.python.org/3.11/c-api/object.html#object

2.4.5 Dimensions section

This section is where you list the dimensions (coordinate axes) your variables are provided on. Each entry should be a key-value pair, where the key is the name your diagnostic uses for that dimension internally, and the value is a list of settings describing that dimension. In order to be unambiguous, all dimensions must specify at least:

Latitude and Longitude

`standard_name`:
Required, string. Must be `"latitude"` and `"longitude"`, respectively.

`units`:
Optional, a CFunit. Units the diagnostic expects the dimension to be in. Currently the framework only supports decimal `degrees_north` and `degrees_east`, respectively.

`range`:
Array (list) of two numbers. Optional. If given, specifies the range of values the diagnostic expects this dimension to take. For example, `"range": [-180, 180]` for longitude will have the first entry of the longitude variable in each data file be near -180 degrees (not exactly -180, because dimension values are cell midpoints), and the last entry near +180 degrees.

`need_bounds`:
Boolean. Optional: assumed `false` if not specified. If `true`, the framework will ensure that bounds are supplied for this dimension, in addition to its midpoint values, following the CF conventions[83]: the `bounds` attribute of this dimension will be set to the name of another netCDF variable containing the bounds information.

`axis`:
String, optional. Assumed to be `Y` and `X` respectively if omitted, or if `standard_name` is `"latitude"` or `"longitude"`. Included here to enable future support for non-lat-lon horizontal coordinates.

Time

`standard_name`:
Required. Must be `"time"`.

`units`:
String. Optional, defaults to "day". Units the diagnostic expects the dimension to be in. Currently the diagnostic only supports time axes of the form "<units> since <reference data>", and the value given here is interpreted in this sense (e.g. settings this to "day" would accommodate a dimension of the form "[decimal] days since 1850-01-01".)

`calendar`:
String, Optional. One of the CF convention calendars[84] or the string `"any"`. **Defaults to "any" if not given**. Calendar convention used by your diagnostic. Only affects the number of days per month.

`need_bounds`:
Boolean. Optional: assumed `false` if not specified. If `true`, the framework will ensure that bounds are supplied for this dimension, in addition to its midpoint values, following the CF conventions[85]: the `bounds` attribute of this dimension will be set to the name of another netCDF variable containing the bounds information.

`axis`:
String, optional. Assumed to be `T` if omitted or provided.

[83] http://cfconventions.org/Data/cf-conventions/cf-conventions-1.8/cf-conventions.html#cell-boundaries
[84] http://cfconventions.org/Data/cf-conventions/cf-conventions-1.8/cf-conventions.html#calendar
[85] http://cfconventions.org/Data/cf-conventions/cf-conventions-1.8/cf-conventions.html#cell-boundaries

Z axis (height/depth, pressure, . . .)

standard_name:
Required, string. Standard name[86] of the variable as defined by the CF conventions[87], or a commonly used synonym as employed in the CMIP6 MIP tables.

units:
Optional, a CFunit. Units the diagnostic expects the dimension to be in. **If not provided, the framework will assume CF convention** canonical units[88].

positive:
String, **required**. Must be `"up"` or `"down"`, according to the CF conventions[89]. A pressure axis is always `"down"` (increasing values are closer to the center of the earth), but this is not set automatically.

need_bounds:
Boolean. Optional: assumed `false` if not specified. If `true`, the framework will ensure that bounds are supplied for this dimension, in addition to its midpoint values, following the CF conventions[90]: the `bounds` attribute of this dimension will be set to the name of another netCDF variable containing the bounds information.

axis:
String, optional. Assumed to be `Z` if omitted or provided.

Other dimensions (wavelength, . . .)

standard_name:
Required, string. Standard name[91] of the variable as defined by the CF conventions[92], or a commonly used synonym as employed in the CMIP6 MIP tables.

units:
Optional, a CFunit. Units the diagnostic expects the dimension to be in. **If not provided, the framework will assume CF convention** canonical units[93].

need_bounds:
Boolean. Optional: assumed `false` if not specified. If `true`, the framework will ensure that bounds are supplied for this dimension, in addition to its midpoint values, following the CF conventions[94]: the `bounds` attribute of this dimension will be set to the name of another netCDF variable containing the bounds information.

2.4.6 Varlist section

Varlist entry example

```
"u500": {
    "standard_name": "eastward_wind",
    "units": "m s-1",
    "realm": "atmos",
    "dimensions" : ["time", "lat", "lon"],
```

(continues on next page)

[86] http://cfconventions.org/Data/cf-standard-names/72/build/cf-standard-name-table.html
[87] http://cfconventions.org/
[88] http://cfconventions.org/Data/cf-standard-names/current/build/cf-standard-name-table.html
[89] http://cfconventions.org/faq.html#vertical_coords_positive_attribute
[90] http://cfconventions.org/Data/cf-conventions/cf-conventions-1.8/cf-conventions.html#cell-boundaries
[91] http://cfconventions.org/Data/cf-standard-names/72/build/cf-standard-name-table.html
[92] http://cfconventions.org/
[93] http://cfconventions.org/Data/cf-standard-names/current/build/cf-standard-name-table.html
[94] http://cfconventions.org/Data/cf-conventions/cf-conventions-1.8/cf-conventions.html#cell-boundaries

(continued from previous page)

```
    "scalar_coordinates": {"plev": 500},
    "requirement": "optional",
    "alternates": ["another_variable_name", "a_third_variable_name"]
}
```

This section is where you list the variables your diagnostic uses. Each entry should be a key-value pair, where the key is the name your diagnostic uses for that variable internally, and the value is a list of settings describing that variable. Most settings here are optional, but the main ones are:

`standard_name`:
The CF standard name[95] for that variable.

`units`:
The units the diagnostic expects the variable to be in (using the syntax of the UDUnits library[96]).

`dimensions`:
List of names of dimensions specified in the "dimensions" section, to specify the coordinate dependence of each variable.

`realm` (if not specified in the *settings* section):
string or list of CMIP modeling realm(s) that the variable belongs to

`modifier`:
String, optional; Descriptor to distinguish variables with identical standard names and different dimensionalities or realms. See modifiers.jsonc[97] for supported modfiers. Open an issue to request the addition of a new modifier to the modifiers.jsonc file, or submit a pull request that includes the new modifier in the modifiers.jsonc file and the necessary POD settings.jsonc file(s).

`requirement`:
String. Optional; assumed `"required"` if not specified. One of three values:

- `"required"`: variable is necessary for the diagnostic's calculations. If the data source doesn't provide the

variable (at the requested frequency, etc., for the user-specified analysis period) the framework will *not* run the diagnostic, but will instead log an error message explaining that the lack of this data was at fault. - `"optional"`: variable will be supplied to the diagnostic if provided by the data source. If not available, the diagnostic will still run, and the `path_variable` for this variable will be set to the empty string. **The diagnostic is responsible for testing the environment variable** for the existence of all optional variables. - `"alternate"`: variable is specified as an alternate source of data for some other variable (see next property). The framework will only query the data source for this variable if it's unable to obtain one of the *other* variables that list it as an alternate.

`alternates`:
Array (list) of strings (e.g., ["A", "B"]), which must be keys (names) of other variables. Optional: if provided, specifies an alternative method for obtaining needed data if this variable isn't provided by the data source.

- If the data source provides this variable (at the requested frequency, etc., for the user-specified

analysis period), this property is ignored. - If this variable isn't available as requested, the framework will query the data source for all of the variables listed in this property. If *all* of the alternate variables are available, the diagnostic will be run; if any are missing it will be skipped. Note that, as currently implemented, only one set of alternates may be given (no "plan B", "plan C", etc.)

`scalar_coordinates`:
optional key-value pair specifying a level to select from a 4-D field. This implements what the CF conventions refer to as "scalar coordinates[98]", with the use case here being the ability to request slices of higher-dimensional

[95] http://cfconventions.org/Data/cf-standard-names/72/build/cf-standard-name-table.html
[96] https://www.unidata.ucar.edu/software/udunits/udunits-2.0.4/udunits2lib.html#Syntax
[97] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/data/modifiers.jsonc
[98] http://cfconventions.org/Data/cf-conventions/cf-conventions-1.8/cf-conventions.html#scalar-coordinate-variables

data. For example, the snippet at the beginning of this section *{"plev": 500}* shows how to request the u component of wind velocity on a 500-mb pressure level.

- *keys* are the key (name) of an entry in the *dimensions* (page 27) section.
- *values* are a single number (integer or floating-point) corresponding to the value of the slice to extract.

Units of this number are taken to be the `units` property of the dimension named as the key.

In order to request multiple slices (e.g. wind velocity on multiple pressure levels, with each level saved to a different file), create one varlist entry per slice.

2.5 POD coding best practices

In this section we describe issues we've seen in POD code that have caused problems in the form of bugs, inefficiencies, or unintended consequences.

2.5.1 All languages

- **PS vs. EPS figures**: Save vector plots as .eps (Encapsulated PostScript), not .ps (regular PostScript).

  Why: Postscript (.ps) is perhaps the most common vector graphics format, and almost all plotting packages are able to output postscript files. Encapsulated Postscript[99] (.eps) includes bounding box information that describes the physical extent of the plot's contents. This is used by the framework to generate bitmap versions of the plots correctly: the framework calls ghostscript[100] for the conversion, and if not provided with a bounding box ghostscript assumes the graphics use an entire sheet of (letter or A4) paper. This can cause plots to be cut off if they extend outside of this region.

  Note that many plotting libraries will set the format of the output file automatically from the filename extension. The framework will process both **.ps* and **.eps* files.

2.5.2 Python: General

- **Whitespace**: Indent python code with four spaces per indent level.

  Why: Python uses indentation to delineate nesting and scope within a program, and indentation that's not done consistently is a syntax error. Using four spaces is not required, but is the generally accepted standard.

  Indentation can be configured in most text editors, or fixed with scripts such as `reindent.py` described here[101]. We recommend using a linter[102] such as `pylint` to find common bugs and syntax errors.

  Beyond this, we don't impose requirements on how your code is formatted, but voluntarily following standard best practices (such as described in PEP8[103] or the Google style guide[104]) will make it easier for you and others to understand your code, find bugs, etc.

- **Filesystem commands**: Use commands in the os[105] and shutil[106] modules to interact with the filesystem, instead of running unix commands using `os.system()`, `commands` (which is deprecated), or `subprocess`.

[99] https://en.wikipedia.org/wiki/Encapsulated_PostScript
[100] https://www.ghostscript.com/
[101] https://stackoverflow.com/q/1024435
[102] https://books.agiliq.com/projects/essential-python-tools/en/latest/linters.html
[103] https://www.python.org/dev/peps/pep-0008/
[104] https://github.com/google/styleguide/blob/gh-pages/pyguide.md
[105] https://docs.python.org/3.11/library/os.html
[106] https://docs.python.org/3.11/library/shutil.html

Why: Hard-coding unix commands makes code less portable. Calling out to a subprocess introduces overhead and makes error handling and logging more difficult. The main reason, however, is that Python already provides these tools in a portable way. Please see the documentation for the os[107] and shutil[108] modules, summarized in this table:

In particular, using os.path.join[109] is more verbose than joining strings but eliminates bugs arising from missing or redundant directory separators.

2.5.3 Python: Arrays

To obtain acceptable performance for numerical computation, people use Python interfaces to optimized, compiled code. NumPy[110] is the standard module for manipulating numerical arrays in Python. xarray[111] sits on top of NumPy and provides a higher-level interface to its functionality; any advice about NumPy applies to it as well.

NumPy and xarray both have extensive documentation and many tutorials, such as:

- NumPy's own basic[112] and intermediate[113] tutorials; xarray's overview[114] and climate and weather examples[115];
- A demonstration[116] of the features of xarray using

Earth science data;

- The 2020 SciPy conference has open-source, interactive tutorials[117] you can work through on your own machine or fully online using Binder[118]. In particular, there are tutorials for NumPy[119] and xarray[120].

- **Eliminate explicit for loops**: Use NumPy/xarray functions instead of writing for loops in Python that loop over the indices of your data array. In particular, nested for loops on multidimensional data should never need to be used.

  Why: For loops in Python are very slow compared to C or Fortran, because Python is an interpreted language. You can think of the NumPy functions as someone writing those for-loops for you in C, and giving you a way to call it as a Python function.

  It's beyond the scope of this document to cover all possible situations, since this is the main use case for NumPy. We refer to the tutorials above for instructions, and to the following blog posts that discuss this specific issue:

  – "Look Ma, no for-loops[121]," by Brad Solomon;

  – "Turn your conditional loops to Numpy vectors[122],"

  by Tirthajyoti Sarkar;

  – "'Vectorized' Operations: Optimized Computations on NumPy Arrays[123]",

[107] https://docs.python.org/3.11/library/os.html
[108] https://docs.python.org/3.11/library/shutil.html
[109] https://docs.python.org/3.10/library/os.path.html?highlight=os%20path#os.path.join
[110] https://numpy.org/doc/stable/index.html
[111] http://xarray.pydata.org/en/stable/index.html
[112] https://numpy.org/doc/stable/user/absolute_beginners.html
[113] https://numpy.org/doc/stable/user/quickstart.html
[114] http://xarray.pydata.org/en/stable/quick-overview.html
[115] http://xarray.pydata.org/en/stable/examples.html
[116] https://rabernat.github.io/research_computing/xarray.html
[117] https://www.scipy2020.scipy.org/tutorial-information
[118] https://mybinder.org/
[119] https://github.com/enthought/Numpy-Tutorial-SciPyConf-2020
[120] https://xarray-contrib.github.io/xarray-tutorial/index.html
[121] https://realpython.com/numpy-array-programming/
[122] https://towardsdatascience.com/data-science-with-python-turn-your-conditional-loops-to-numpy-vectors-9484ff9c622e
[123] https://www.pythonlikeyoumeanit.com/Module3_IntroducingNumpy/VectorizedOperations.html

part of "Python like you mean it[124]," a free resource by Ryan Soklaski.

- **Use xarray with netCDF data**:

  Why: This is xarray's use case. You can think of NumPy as implementing multidimensional matrices in the fully general, mathematical sense, and xarray providing the specialization to the case where the matrix contains data on a lat-lon-time-(etc.) grid.

  xarray lets you refer to your data with human-readable labels such as 'latitude,' rather than having to remember that that's the second dimension of your array. This bookkeeping is essential when writing code for the MDTF framework, when your POD will be run on data from models you haven't been able to test on.

  In particular, xarray provides seamless support for time axes[125], with support[126] for all CF convention calendars through the `cftime` library. You can, eg, subset a range of data between two dates without having to manually convert those dates to array indices.

  See the xarray tutorials linked above for more examples of xarray's features.

- **Memory use and views vs. copies**: Use scalar indexing and

  slices[Page 32, 127]

  (index specifications of the form *start_index*:*stop_index*:*stride*) to get subsets of arrays whenever possible, and only use advanced indexing[128] features (indexing arrays with other arrays) when necessary.

  Why: When advanced indexing is used, NumPy will need to create a new copy of the array in memory, which can hurt performance if the array contains a large amount of data. By contrast, slicing or basic indexing is done in-place, without allocating a new array: the NumPy documentation calls this a "view."

  Note that array slices are native Python objects[129], so you can define a slice in a different place from the array you intend to use it on. Both NumPy and xarray arrays recognize slice objects.

  This is easier to understand if you think about NumPy as a wrapper around C-like functions: array indexing in C is implemented with pointer arithmetic, since the array is implemented as a contiguous block of memory. An array slice is just a pointer to the same block of memory, but with different offsets. More complex indexing isn't guaranteed to follow a regular pattern, so NumPy needs to copy the requested data in that case.

  See the following references for more information:

  - The NumPy documentation[130] on indexing;
  - "Numpy Views vs Copies: Avoiding Costly Mistakes[131]," by Jessica Yung;
  - "How can I tell if NumPy creates a view or a copy?[132]" on stackoverflow.

- **MaskedArrays instead of NaNs or sentinel values**: Use NumPy's MaskedArrays[133] for data that may contain missing or invalid values, instead of setting those entries to NaN or a sentinel value.

[124] https://www.pythonlikeyoumeanit.com/
[125] http://xarray.pydata.org/en/stable/time-series.html
[126] http://xarray.pydata.org/en/stable/weather-climate.html
[127] https://numpy.org/doc/stable/reference/arrays.indexing.html#basic-slicing-and-indexing
[128] https://numpy.org/doc/stable/reference/arrays.indexing.html#advanced-indexing
[129] https://docs.python.org/3.7/library/functions.html?highlight=slice#slice
[130] https://numpy.org/doc/stable/reference/arrays.indexing.html
[131] https://www.jessicayung.com/numpy-views-vs-copies-avoiding-costly-mistakes/
[132] https://stackoverflow.com/questions/11524664/how-can-i-tell-if-numpy-creates-a-view-or-a-copy
[133] https://numpy.org/doc/stable/reference/maskedarray.generic.html

Why: One sometimes encounters code which sets array entries to fixed "sentinel values" (such as 1.0e+20 or NaN[134]) to indicate missing or invalid data. This is a dangerous and error-prone practice, since it's frequently not possible to detect if the invalid entries are being used by mistake. For example, computing the variance of a timeseries with missing elements set to 1e+20 will either result in a floating-point overflow, or return zero.

NumPy provides a better solution in the form of MaskedArrays[135], which behave identically to regular arrays but carry an extra boolean mask to indicate valid/invalid status. All the NumPy mathematical functions will automatically use this mask for error propagation. For example[136], trying to divide an array element by zero or taking the square root of a negative element will mask it off, indicating that the value is invalid: you don't need to remember to do these sorts of checks explicitly.

2.5.4 Python: Plotting

- **Use the 'Agg' backend when testing your POD**: For reproducibility, set the shell environment variable

`MPLBACKEND` to `Agg` when testing your POD outside of the framework.

Why: Matplotlib can use a variety of backends[137]: interfaces to low-level graphics libraries. Some of these are platform-dependent, or require additional libraries that the MDTF framework doesn't install. In order to achieve cross-platform portability and reproducibility, the framework specifies the `'Agg'` non-interactive (ie, writing files only) backend for all PODs, by setting the `MPLBACKEND` environment variable.

When developing your POD, you'll want an interactive backend – for example, this is automatically set up for you in a Jupyter notebook. When it comes to testing your POD outside of the framework, however, you should be aware of this backend difference.

2.6 Git-based development workflow

2.6.1 Steps for brand new users:

1. Fork the MDTF-diagnostics branch to your GitHub account (*Creating a fork of the MDTF-diagnostics repository* (page 34))
2. Clone (*Cloning a repository onto your machine* (page 34)) your fork of the MDTF-diagnostics repository (repo) to your local machine (if you are not using the web interface for development)
3. Check out a new branch from the local main branch (*Working on a brand new POD* (page 35))
4. Start coding
5. Commit the changes in your POD branch (*Working on a brand new POD* (page 35))
6. Push the changes to the copy of the POD branch on your remote fork (*Working on a brand new POD* (page 35))
7. Repeat steps 4–6 until you are finished working
8. Submit a pull request to the NOAA-GFDL repo for review (*Submitting Pull Requests* (page 35)).

[134] https://en.wikipedia.org/wiki/NaN
[135] https://numpy.org/doc/stable/reference/maskedarray.html
[136] https://numpy.org/doc/stable/reference/maskedarray.generic.html#numerical-operations
[137] https://matplotlib.org/tutorials/introductory/usage.html#backends

2.6.2 Steps for users continuing work on an existing POD branch

1. Create a backup copy of the MDTF-Diagnostics repo on your local machine
2. Pull in updates from the NOAA-GFDL/main branch to the main branch in your remote repo (*Updating your remote and local main branches* (page 36))
3. Pull in updates from the main branch in your remote fork into the main branch in your local repo (*Updating your remote and local main branches* (page 36))
4. Sync your POD branch in your local repository with the local main branch using an interactive rebase (*Updating your POD branch by rebasing it onto the main branch (a bit more difficult than merging, but cleaner)* (page 37)) or merge (*Updating your POD branch by merging in changes from the main branch* (page 38)). Be sure to make a backup copy of of your local *MDTF-diagnostics* repo first, and test your branch after rebasing/merging as described in the linked instructions before proceeding to the next step.
5. Continue working on your POD branch
6. Commit the changes in your POD branch
7. Push the changes to the copy of the POD branch in your remote fork (*Pushing to your remote POD branch on your fork* (page 35))
8. Submit a pull request (PR) to NOAA-GFDL/main branch when your code is ready for review (*Submitting Pull Requests* (page 35))

2.6.3 Creating a fork of the MDTF-diagnostics repository

- If you have no prior experience with GitHub[138], create an account first.
- Enable multifactor authentication[139] on your Github account
- Create a *fork* of the project by clicking the `Fork` button in the upper-right corner of NOAA's MDTF GitHub page[140]. This will create a copy (also known as *repository*, or simply *repo*) in your own GitHub account which you have full control over.

2.6.4 Cloning a repository onto your machine

Before following the instructions below, make sure that a) you've created a fork of the project, and b) the `git` command is available on your machine (installation instructions[141]).

- *Clone* your fork onto your computer: `git clone git@github.com:<your_github_account>/MDTF-diagnostics.git`. This not only downloads the files, but due to the magic of git also gives you the full commit history of all branches.
- Enter the project directory: `cd MDTF-diagnostics`.
- Git knows about your fork, but you need to tell it about NOAA's repo if you wish to contribute changes back to the code base. To do this, type `git remote add upstream git@github.com:NOAA-GFDL/MDTF-diagnostics.git`.

Now you have two remote repos: `origin`, your GitHub fork which you can read and write to, and `upstream`, NOAA's code base which you can only read from.

[138] https://github.com/
[139] https://docs.github.com/en/authentication/securing-your-account-with-two-factor-authentication-2fa/accessing-github-using-two-factor-authentication
[140] https://github.com/NOAA-GFDL/MDTF-diagnostics
[141] https://git-scm.com/download/

Another approach is to create a local repo on your machine and manage the code using the `git` command in a terminal. In the interests of making things self-contained, the rest of this section gives brief step-by-step instructions on git for interested developers.

2.6.5 Working on a brand new POD

Developers can either clone the MDTF-diagnostics repo to their computer, or manage the MDTF package using the GitHub webpage interface. Whichever method you choose, remember to create your [POD branch name] branch from the main branch, not the main branch. Since developers commonly work on their own machines, this manual provides command line instructions.

1. Check out a branch for your POD

```
git checkout -b [POD branch name]
```

2. Write code, add files, etc…

3. Add the files you created and/or modified to the staging area

```
git add [file 1]
git add [file 2]
...
```

4. Commit your changes, including a brief description

```
git commit -m "description of my changes"
```

5. Push the updates to your remote repository

```
git push -u origin [POD branch name]
```

2.6.6 Pushing to your remote POD branch on your fork

When you are ready to push your updates to the remote POD branch on your fork

1. Type `git status` to list the file(s) that have been updated
2. Repeat steps 3–5 of section (*Working on a brand new POD* (page 35))

2.6.7 Submitting Pull Requests

The pull request (PR) for your branch is your proposal to the maintainers to incorporate your POD into NOAA's repo. Your changes will not affect the official NOAA's repo until the PR is accepted by the lead-team programmer. Note that if any buttons are missing, try `CRTL + +` or `CRTL + -` to adjust the webpage font size so the missing buttons may magically appear.

To submit a PR :

1. Click the *Contribute* link on the main page of your MDTF-diagnostics fork and click the *Open Pull Request* button
2. Verify that your fork is set as the **base** repository, and *main* is set as the **base branch**, that *NOAA-GFDL* is set as the **head repository**, and *main* is set as the **head** branch
3. Click the *Create Pull Request* button, add a brief description to the PR header, and go through the checklist to ensure that your code meets that baseline requirements for review

4. Click the *Create Pull Request* button (now in the lower left corner of the message box)

Note that you can submit a Draft Pull Request if you want to run the code through the CI, but are not ready for a full review by the framework team. Starting from step 3. above

1. Click the arrow on the right edge of the *Create Pull Request* button and select *Create draft pull request* from the dropdown menu.
2. Continue pushing changes to your POD branch until you are ready for a review (the PR will update automatically)
3. When you are ready for review, navigate to the NOAA-GFDL/MDTF-Diagnostics *Pull requests*[142] page, and click on your PR
4. Scroll down to the header that states "this pull request is still a work in progress", and click the *ready for review* button to move the PR out of *draft* mode

2.6.8 Updating your remote and local main branches

Method 1: Web interface+command line

See the MDTF Best Practices Overview[143] presentation for instructions with figures.

1. Click the *Fetch Upstream* link on the main page of your MDTF-diagnostics fork, then click the *Open Pull Request* button
2. Verify that your fork is set as the **base** repository, and *main* is set as the **base branch**, that *NOAA-GFDL* is set as the **head repository**, and *main* is set as the **head** branch
3. Create a title for your PR, add a description if you want, then click *Create pull request*
4. Click **Merge pull request**

Your remote main branch is now up-to-date with the NOAA-GFDL/main branch.

5. On your machine, open a terminal and check out the main branch

```
git checkout main
```

6. Fetch the updates to the main branch from your remote fork

```
git fetch
```

7. Pull in the updates from the remote main branch.

```
git pull
```

Your local main branch is now up-to-date with the NOAA-GFDL/main branch.

[142] https://github.com/NOAA-GFDL/MDTF-diagnostics/pulls
[143] https://docs.google.com/presentation/d/18jbi50vC9X89vFbL0W1Ska1dKuW_yWY51SomWx_ahYE/edit?usp=sharing

Method 2: Command line only

This method requires adding the *NOAA-GFDL/MDTF-diagnostics* repo to the *.git/config* file in your local repo, and is described in the GitHub discussion post Working with multiple remote repositories in your git config file[144].

2.6.9 Updating your POD branch by rebasing it onto the main branch (a bit more difficult than merging, but cleaner)

Rebasing is procedure to integrate the changes from one branch into another branch. `git rebase` differs from `git merge` in that it reorders the commit history so that commits from the branch that is being updated are moved to the *tip* of the branch. This makes it easier to isolate changes in the POD branch, and usually results in fewer merge conflicts when the POD branch is merged into the main branch. 1. Create a backup copy of your MDTF-diagnostics repo on your local machine

2. Update the local and remote main branches on your fork as described in *Updating your remote and local main branches* (page 36), then check out your POD branch

```
git checkout [POD branch name]
```

and launch an interactive rebase of your branch onto the main branch:: git rebase -i main 3. Your text editor will open in the terminal (Vim by default) and display your commit hashes with the oldest commit at the top

```
pick 39n3b42 oldest commit
pick 320cnyn older commit
pick 20ac93c newest commit
```

You may squash commits by replacing *pick* with *squash* for the commit(s) that are newer than the commit you want to combine with (i.e., the commits below the target commit). For example

```
pick 39n3b42 oldest commit
squash 320cnyn older commit
pick 20ac93c newest commit
```

combines commit 320cnyn with commit 29n3b42, while

```
pick 39n3b42 oldest commit
squash 320cnyn older commit
squash 20ac93c newest commit
```

combines 20ac93c and 320cnyn with 39n3b42.

Note that squashing commits is not required. However, doing so creates a more streamlined commit history.

4. Once you're done squashing commits (if you chose to do so), save your changes and close the editor `ESC + SHIFT + wq` to save and quit in Vim), and the rebase will launch. If the rebase stops because there are merge conflicts and resolve the conflicts. To show the files with merge conflicts, type

```
git status
```

This will show files with a message that there are merge conflicts, or that a file has been added/deleted by only one of the branches. Open the files in an editor, resolve the conflicts, then add edited (or remove deleted) files to the staging area

[144] https://github.com/NOAA-GFDL/MDTF-diagnostics/discussions/96

```
git add file1
git add file2
...
git rm file3
```

5. Next, continue the rebase

```
git rebase --continue
```

The editor will open with the modified commit history. Simply save the changes and close the editor (ESC+SHIFT+wq), and the rebase will continue. If the rebase stops with errors, repeat the merge conflict resolution process, add/remove the files to staging area, type `git rebase --continue`, and proceed.

If you have not updated your branch in a long time, you'll likely find that you have to keep fixing the same conflicts over and over again (every time your commits collide with the commits on the main branch). This is why we strongly advise POD developers to pull updates into their forks and rebase their branches onto the main branch frequently.

Note that if you want to stop the rebase at any time and revert to the original state of your branch, type

```
git rebase --abort
```

6. Once the rebase has completed, push your changes to the remote copy of your branch

```
git push -u origin [POD branch name] --force
```

The `--force` option is necessary because rebasing modified the commit history.

7. Now that your branch is up-to-date, write your code!

2.6.10 Updating your POD branch by merging in changes from the main branch

1. Create a backup copy of your repo on your machine.
2. Update the local and remote main branches on your fork as described in *Updating your remote and local main branches* (page 36).

3. Check out your POD branch, and merge the main branch into your POD branch

```
git checkout [POD branch name]
git merge main
```

4. Resolve any conflicts that occur from the merge

5. Add the updated files to the staging area

```
git add file1
git add file2
...
```

6. Push the branch updates to your remote fork

```
git push -u origin [POD branch name]
```

Reverting commits

If you want to revert to the commit(s) before you pulled in updates:

1. Find the commit hash(es) with the updates, in your git log

```
git log
```

or consult the commit log in the web interface

2. Revert each commit in order from newest to oldest

```
git revert <newer commit hash>
git revert <older commit hash>
```

3. Push the updates to the remote branch

```
git push origin [POD branch name]
```

2.6.11 Set up SSH with GitHub

- You have to generate an SSH key[145] and add it[146] to your GitHub account. This will save you from having to re-enter your GitHub username and password every time you interact with their servers.
- When generating the SSH key, you'll be asked to pick a *passphrase* (i.e., password).
- The following instructions assume you've generated an SSH key. If you're using manual authentication instead, replace the "`git@github.com:`" addresses in what follows with "`https://github.com/`".

2.6.12 Some online git resources

If you are new to git and unfamiliar with many of the terminologies, Dangit, Git?![147] provides solutions *in plain English* to many common mistakes people have made.

There are many comprehensive online git tutorials, such as:

- The official git tutorial[148].
- A more verbose introduction[149] to the ideas behind git and version control.
- A still more detailed walkthrough[150], assuming no prior knowledge.

[145] https://help.github.com/en/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent
[146] https://help.github.com/en/articles/adding-a-new-ssh-key-to-your-github-account
[147] https://dangitgit.com/
[148] https://git-scm.com/docs/gittutorial
[149] https://www.atlassian.com/git/tutorials/what-is-version-control
[150] http://swcarpentry.github.io/git-novice/

2.6.13 Git Tips and Tricks

- If you are unfamiliar with git and want to practice with the commands listed here, we recommend you to create an additional POD branch just for this. Remember: your changes will not affect NOAA's repo until you've submitted a pull request through the GitHub webpage and accepted by the lead-team programmer.
- GUI applications can be helpful when trying to resolve merge conflicts.Git packages for IDEs such as VSCode, Pycharm, and Eclipse often include tools for merge conflict resolution. You can also install free versions of merge-conflict tools like P4merge[151] and Sublime merge[152].
- If you encounter problems during practice, you can first try looking for *plain English* instructions to fix the situation at Dangit, Git?![153].
- A useful command is `git status` to remind you what branch you're on and changes you've made (but have not committed yet).
- `git branch -a` lists all branches with * indicating the branch you're on.
- Push your changes to your remote fork often (at least daily) even if your changes aren't "clean", or you are in the middle of a task. Your commit history does not need to look like a polished document, and nobody is judging your coding prowess by your development branch. Frequently pushing to your remote branch ensures that you have an easily accessible recent snapshot of your code in the event that your system goes down, or you go crazy with `rm -f *`.
- **A commit creates a snapshot of the code into the history in your local repo.**
  - The snapshot will exist until you intentionally delete it (after confirming a warning message). You can always revert to a previous snapshot.
  - Don't commit code that you know is buggy or non-functional!
  - You'll be asked to enter a commit message. Good commit messages are key to making the project's history useful.
  - Write in *present tense* describing what the commit, when applied, does to the code – not what you did to the code.
  - Messages should start with a brief, one-line summary, less than 80 characters. If this is too short, you may want to consider entering your changes as multiple commits.
- Good commit messages are key to making the project's history useful. To make this easier, instead of using the `-m` flag, To provide further information, add a blank line after the summary and wrap text to 72 columns if your editor supports it (this makes things display nicer on some tools). Here's an example[154].
- To configure git to launch your text editor of choice: `git config --global core.editor "<command string to launch your editor>"`.
- To set your email: `git config --global user.email "myemail@somedomain.com"` You can use the masked email Github provides if you don't want your work email included in the commit log message. The masked email address is located in the *Primary email address* section under Settings > emails.
- When the POD branch is no longer needed, delete the branch locally with `git branch -d [POD branch name]`. If you pushed the POD branch to your fork, you can delete it remotely with `git push --delete origin [POD branch name]`. - Remember that branches in git are just pointers to a particular commit, so by deleting a branch you *don't* lose

  any history.

[151] https://www.perforce.com/products/helix-core-apps/merge-diff-tool-p4merge
[152] https://www.sublimemerge.com/
[153] https://dangitgit.com/
[154] https://github.com/NOAA-GFDL/MDTF-diagnostics/commit/225b29f30872b60621a5f1c55a9f75bbcf192e0b

- If you want to let others work on your POD, push the POD branch to your GitHub fork with `git push -u origin [POD branch name`.
- For additional ways to undo changes in your branch, see How to undo (almost) anything with Git[155].

2.7 Development Cheatsheet

2.7.1 Creating and submitting a POD

1. Prepare for implementation
   - Run the unmodified MDTF-diagnostics package to make sure that your conda installation, directory structure, etc… are set up properly
   - Modify the conda environment to work for your POD by adding a configuration file `MDTF_diagnostics/src/conda/env_[YOUR POD NAME].yml` with any new required modules. Be sure to re-run `MDTF-diagnostics/src/conda/conda_env_setup.sh` to install your POD's environment if it requires a separate YAML file with additional modules.
   - Name your POD, make a directory for your POD in MDTF-diagnostics/diagnostics, and move your code to your POD directory
   - `cp` your observational data to `MDTF_diagnostics/../inputdata/obs_data/[YOUR POD NAME]`
2. Link your POD code into the framework - Modify your POD's driver script (e.g, `driver.py`) to interface with your code - Modify pod's `settings.jsonc` to specify variables that will be passed to the framework - Modify your code to use `ENV_VARS` provided by the framework (see the *Notes* for descriptions of the available

   environment variables)

   - **Input files:**
     - model input data: specified in an ESM-intake catalog
     - observational input data: `MDTF-diagnostics/../inputdata/obs_data/[POD name]`
     - You may re-define input data locations in the `OBS_DATA_ROOT` setting in your runtime configuration file (or whatever the name of your runtime settings jsonc file is).
   - **Working files:**
     - `${WORK_DIR}` is a framework environment variable defining the working directory. It is set to

     `MDTF-diagnostics/../wkdir` by default. - `${WORK_DIR}` contains temporary files and logs. - You can modify `${WORK_DIR}` by changing "WORK_DIR" to the desired location in

     `templates/runtime.[jsonc |yml}`
   - **Output files:**
     - **POD output files are written to the following locations by the framework:**
       - Postscript files: `${WORK_DIR}/MDTF_output[.v#]/[POD NAME]/[model, obs]/PS`
       - Other files, including PNG plots: `${WORK_DIR}/MDTF_output[.v#]/[POD NAME]/[model,obs]`

[155] https://github.blog/2015-06-08-how-to-undo-almost-anything-with-git/

  - Set the "OUTPUT_DIR" option in default_tests.jsonc to write output files to a different location; "OUTPUT_DIR" defaults to "WORK_DIR" if it is not defined.
  - **Output figure locations:**
    * PNG files should be placed directly in `$WORK_DIR/obs/` and `$WORK_DIR/model/`
    * If a POD chooses to save vector-format figures, they should be written into the `$WORK_DIR/MDTF_output[.v#]/[POD_NAME]/obs/PS` and `$WORK_DIR/MDTF_output[.v#]/[POD_NAME]/model/PS` directories. Files in these locations will be converted by the framework to PNG, so use those names in the html file.
    * If a POD uses matplotlib, it is recommended to write as figures as EPS instead of PS because of potential bugs
- Modify html files to point to the figure names

3. Place your documentation in `MDTF-diagnostics/diagnostics/[YOUR POD NAME]/docs`
4. Test your code with the framework
5. Submit a Pull Request for your POD using the GitHub website

2.7.2 Notes:

- **Make sure that WORK_DIR and OUTPUT_DIR have enough space to hold data for your POD(s) AND any PODs included in the package.**
- **Defining POD variables**
  - Add variables to the `varlist` block in the `MDTF-diagnostics/diagnostics/[POD name]/settings.jsonc` and define the following:
    * the variable name: the short name that will generate the corresponding `${ENV_VAR}` (e.g., "zg500" generates the `${ENV_VAR}` "zg500_var")
    * the standard name with a corresponding entry in the appropriate fieldlist file(s)
    * variable units
    * variable dimensions (e.g., [time, lat, lon])
    * variable realm (e.g., atmos, ocean ice, land)
    * scalar coordinates for variables defined on a specific atmospheric pressure level (e.g. `{"lev": 250}` for a field on the 250-hPa p level).
  - If your variable is not in the necessary fieldlist file(s), add them to the file(s), or open an issue on GitHub requesting that the framework team add them. Once the files are updated, merge the changes from the main branch into your POD branch.
  - Note that the variable name and the standard name must be unique fieldlist entries
- **Environment variables**
  - To define an environment variable specific to your POD, add a `"pod_env_vars"` block to the `"settings"` block in your POD's `settings.jsonc` file and define the desired variables
  - Reference an environment variable associated with a specific case in Python by calling `os.environ[case_env_file]`, reading the file contents into a Python dictionary, and getting value associated with the first case (assuming variable names and coordinates are identical for each case), e.g. `tas_var = [case['tas_var'] for case in case_list.values()][0]`. See `example_multicase.py` for more information.

- NCL code can reference environment variables by calling `getenv("VARIABLE NAME")`
- **Framework-specific environment variables include:**
    - **case_env_file: path to yaml file with case-specific environment variables:**
        - DATA_CATALOG: path to the ESM-intake catalog with model input files and metadata
        - CASELIST: list of case identfiers corresponding to each model simulation
        - startdate: string in yyyymmdd or yyyymmddHHMMSS specifying the start date of the analysis period
        - enddate: string in yyyymmdd or yyyymmddHHMMSS specifying the end date of the analysis period
        - [variable id]_var: environment variable name assigned to variable
        - time_coord: time coordinate
        - lat_coord: latitude coordinate
        - lon_coord: longitude coordinate
    - OBS_DATA: path to the top-level directory containing any observational or reference data for your POD
    - WORK_DIR: path to the POD working directory

CHAPTER

THREE

DIAGNOSTICS REFERENCE

3.1 Example Diagnostic Documentation

Last update: 5/06/2020

This is an example document that you can use as a template for your diagnostics' own documentation: it describes what information you should provide in each section. For example, if this were a real POD, you'd place a one-paragraph synopsis of your diagnostic here (like an abstract).

It also serves as an example of the RestructuredText (ReST, .rst) format used to generate this page: compare this output with the input source file[156]. The easiest way to get started is to copy the source file into the online editor at https://livesphinx.herokuapp.com/ and experiment.

3.1.1 Version & Contact info

Here you should describe who contributed to the diagnostic, and who should be contacted for further information:

- Version/revision information: version 1 (5/06/2020)
- PI (name, affiliation, email)
- Developer/point of contact (name, affiliation, email)
- Other contributors

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt). Unless you've distributed your script elsewhere, you don't need to change this.

3.1.2 Functionality

In this section you should summarize the stages of the calculations your diagnostic performs, and how they translate to the individual source code files provided in your submission. This will, e.g., let maintainers fixing a bug or people with questions about how your code works know where to look.

[156] https://raw.githubusercontent.com/NOAA-GFDL/MDTF-diagnostics/develop/diagnostics/example/doc/example.rst

3.1.3 Required programming language and libraries

In this section you should summarize the programming languages and third-party libraries used by your diagnostic. You also provide this information in the `settings.jsonc` file, but here you can give helpful comments to human maintainers (eg, "We need at least version 1.5 of this library because we call this function.")

3.1.4 Required model output variables

In this section you should describe each variable in the input data your diagnostic uses. You also need to provide this in the `settings.jsonc` file, but here you should go into detail on the assumptions your diagnostic makes about the structure of the data.

3.1.5 References

Here you should cite the journal articles providing the scientific basis for your diagnostic. To keep the documentation format used in version 2.0 of the framework, we list references "manually" with the following command:

```
.. _ref-Maloney:

1. E. D. Maloney et al. (2019): Process-Oriented Evaluation of Climate
and Weather Forecasting Models. *BAMS*, **100** (9), 1665-1686,
`doi:10.1175/BAMS-D-18-0042.1 <https://doi.org/10.1175/BAMS-D-18-0042.1>`__.
```

which produces

1. E. D. Maloney et al. (2019): Process-Oriented Evaluation of Climate and Weather Forecasting Models. *BAMS*, **100** (9), 1665–1686, doi:10.1175/BAMS-D-18-0042.1[157].

which can be cited in text as `:ref:`a hyperlink <reference tag>``, which gives *a hyperlink* (page 46) to the location of the reference on the page. Because references are split between this section and the following "More about this diagnostic" section, unfortunately you'll have to number references manually.

We don't enforce any particular bibliographic style, but please provide a hyperlink to the article's DOI for ease of online access. Hyperlinks are written as `` `link text <URL>`__ `` (text and url enclosed in backticks, followed by two underscores).

3.1.6 More about this diagnostic

In this section, you can go into more detail on the science behind your diagnostic, for example, by copying in relevant text articles you've written using th It's especially helpful if you're able to teach users how to use your diagnostic's output, by showing how to interpret example plots.

Instead of doing that here, we provide more examples of RestructuredText syntax that you can customize as needed.

As mentioned above, we recommend the online editor at https://livesphinx.herokuapp.com/, which gives immediate feedback and has support for sphinx-specific commands.

Here's an introduction[158] to the RestructuredText format, a quick reference[159], and a syntax comparison[160] to other text formats you may be familiar with.

[157] https://doi.org/10.1175/BAMS-D-18-0042.1
[158] http://docutils.sourceforge.net/docs/user/rst/quickstart.html
[159] http://docutils.sourceforge.net/docs/user/rst/quickref.html
[160] http://hyperpolyglot.org/lightweight-markup

Links to external sites

URLs written out in the text are linked automatically: https://ncar.ucar.edu/.

To use custom text for the link, use the syntax `` `link text <https://www.noaa.gov/>`__ `` (text and url enclosed in backticks, followed by two underscores). This produces link text[161].

More references and citations

Here's another reference:

```
.. _ref-Charney:

2. Charney, Jule; Fjørtoft, Ragnar; von Neumann, John (1950). Numerical
Integration of the Barotropic Vorticity Equation. *Tellus* **2** (4) 237-254,
`doi:10.3402/tellusa.v2i4.8607 <https://doi.org/10.3402/tellusa.v2i4.8607>`__.
```

2. Charney, Jule; Fjørtoft, Ragnar; von Neumann, John (1950). Numerical Integration of the Barotropic Vorticity Equation. *Tellus* **2** (4) 237–254, doi:10.3402/tellusa.v2i4.8607[162].

Here's an example of citing these references:

```
:ref:`Maloney et. al., 2019 <ref-Maloney>`,
:ref:`Charney, Fjørtoft and von Neumann, 1950 <ref-Charney>`
```

produces *Maloney et. al., 2019* (page 46), *Charney, Fjørtoft and von Neumann, 1950* (page 47).

Figures

Images **must** be provided in either .png or .jpeg formats in order to be displayed properly in both the html and pdf output.

Here's the syntax for including a figure in the document:

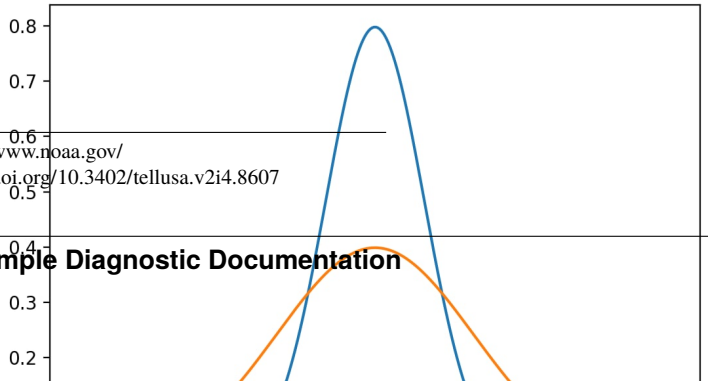
```
.. _my-figure-tag: [only needed for linking to figures]

.. figure:: [path to image file, relative to the source.rst file]
   :align: left
   :width: 75 % [these both need to be indented by three spaces]

   Paragraphs or other text following the figure that are indented by three
   spaces are treated as a caption/legend, eg:

   - red line: a Gaussian
   - blue line: another Gaussian
```

which produces



The tag lets you refer to figures in the text, e.g. `:ref:`Figure 1 <my-figure-tag>`` → *Figure 1* (page 47).

[161] https://www.noaa.gov/
[162] https://doi.org/10.3402/tellusa.v2i4.8607

Equations

Accented and Greek letters can be written directly using Unicode: é, . (Make sure your text editor is saving the file in UTF-8 encoding).

Use the following syntax for superscripts and subscripts in-line text:

```
W m\ :sup:`-2`\
↪ ; CO\ :sub:`2`\ .
```

which produces: W m^{-2}; CO_2. Note one space is needed after both forward slashes in the input; these spaces are not included in the output.

Equations can be written using standard latex[163] (PDF link) syntax. Short equations in-line with the text can be written as `:math:`f = 2 \Omega \sin \phi`` → $f = 2\Omega \sin \phi$.

Longer display equations can be written as follows. Note that a blank line is needed after the `.. math::` heading and after each equation, with the exception of aligned equations.

```
.. math::

   \frac{D \mathbf{u}_g}{Dt} + f_0 \hat{\mathbf{k}} \times \mathbf{u}_a &= 0; \\
   \frac{Dh}{Dt} + f \nabla_z \cdot \mathbf{u}_a &= 0,

   \text{where } \mathbf{u}_g = \frac{g}{f_0} \hat{\mathbf{k}} \times \nabla_z h.
```

which produces:

$$\frac{D\mathbf{u}_g}{Dt} + f_0\hat{\mathbf{k}} \times \mathbf{u}_a = 0;$$
$$\frac{Dh}{Dt} + f\nabla_z \cdot \mathbf{u}_a = 0,$$
$$\text{where } \mathbf{u}_g = \frac{g}{f_0}\hat{\mathbf{k}} \times \nabla_z h.$$

The editor at https://livesphinx.herokuapp.com/ can have issues formatting complicated equations, so you may want to check its output with a latex-specific editor, such as overleaf[164] or other equation editors[165].

[163] https://www.reed.edu/academic_support/pdfs/qskills/latexcheatsheet.pdf
[164] https://www.overleaf.com/
[165] https://www.codecogs.com/latex/eqneditor.php

3.2 Rich Neale's Blocking Diagnostic Documentation

Evaluate blocking frequency by season as determined by the meridional gradient above a threshold value of daily 500mb height following *D'Andrea et al (1998)* (page 52) and a 2D variant of the above where lat and lon daily 500-mb thresholds are considered *Davini et al* (page 52)

3.2.1 Version & Contact info

Version 2.0, implemented in MDTF v3.0beta3
PI: Rich Neale, NCAR, rneale@ucar.edu
Contact: Dani Coleman, NCAR, bundy@ucar.edu

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt)

3.2.2 Functionality

Required programming language and libraries

- Python 3
- NCL (usual NCAR/CESM packages: contributed, gsn_code, gsn_csm)

Required model output variables

Geopotential height at 500 milibars or 3D field with air pressure available

Settings/Environment Variables

There are two sets of environment variables that can be set in the input jsonc file:

1) What to compare input case(s) against:
    - observational products from ERAI & MERRA. Can be turned on or off together. Default: On Years used can be controlled individually
    - three CESM 5-member ensembles (CAM3, CAM4, CAM5) which can be turned on or off individually. Default: CAM5 only. (Note that the code runs with these but the html file does not yet respond to this. Therefore the POD will fail to link the figures if non-default options are used)

Variable name	Default

"MDTF_BLOCKING_OBS" : "True", //both ERA & MERRA (not possible to choose individually yet) "MDTF_BLOCKING_OBS_USE_CASE_YEARS": "False", //if False, must set ERA/MERRA/CAM5 FIRST/LAST-YRS below "MDTF_BLOCKING_OBS_ERA_FIRSTYR" : 2010, "MDTF_BLOCKING_OBS_ERA_LASTYR" : 2014, "MDTF_BLOCKING_OBS_MERRA_FIRSTYR" : 2009, "MDTF_BLOCKING_OBS_MERRA_LASTYR" : 2011, "MDTF_BLOCKING_OBS_CAM5_FIRSTYR" : 1979, "MDTF_BLOCKING_OBS_CAM5_LASTYR" : 2007, "MDTF_BLOCKING_CAM3"

: "False", //if True, will run but doesn't show on webpage "MDTF_BLOCKING_CAM4" : "False", //if True, will run but doesn't show on webpage "MDTF_BLOCKING_CAM5" : "True", "MDTF_BLOCKING_READ_DIGESTED" : "True", // if True, files must be available "MDTF_BLOCKING_WRITE_DIGESTED" : "False", // if True, writes out case data as digested. requires READ_DIGESTED = False "MDTF_BLOCKING_WRITE_DIGESTED_DIR": "", // default output directory for digested data "MDTF_BLOCKING_DEBUG" : "False" // reduces number of ensemble members to 2 for quicker execution

REMOVED:
"MDTF_BLOCKING_COMPARE_LONG_YEARS" use instead the more detailed variables above

2) Variables to control reading or writing of digested data

- `MDTF_BLOCKING_READ_DIGESTED (Default True)` If True, the POD looks for digested data for the obs & CAM ensembles Digested means the data has been processed by this POD into the *block_time* variable name and format (see `MDTF_BLOCKING_WRITE_DIGESTED` below) `(MDTF_BLOCKING_READ_DIGESTED = False)` If False, the POD looks for raw data (Z500(time,lat,lon)).

  The POD is not currently capable of running with a mix of digested and undigested (other than the undigested MDTF input case). If the user desires this, it is recommended to run only the components that are not digested, and write out digested. Then re-running with MDTF_BLOCKING_READ_DIGESTED will work.

  It is not yet possible to run the MDTF input case with digested data, although it is possible to write it out.

  OBS/CAM file name & dir expectations:
  Digested: obs_data/blocking/ERAI/ERAI.z500.day. *digested* .nc
  Undigested: obs_data/blocking/*undigested*/ERAI/ERAI.z500.day.nc

- `MDTF_BLOCKING_WRITE_DIGESTED (Default False)` If True, whatever data was read in raw will be written out digested. This is how a user can make a new digested file, whether obs, a new case or ensemble, or a previous MDTF input case to compare against.

3.2.3 Code Overview

The POD is called by a python wrapper script, `blocking_neale.py` which calls `blocking.ncl`.

`blocking.ncl` summary:

1. Read all case data from environmental variables and store it in `all_cases` structure

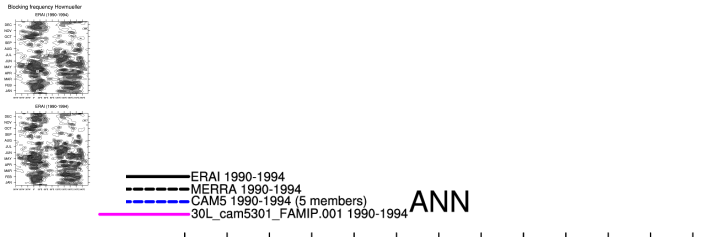
   `all_cases = blocking_get_all_case_info()` is a data structure defined in blocking_funcs.ncl which contains arrays of length number of cases (mdtf case + obs cases + comparison models) and contains the following components:

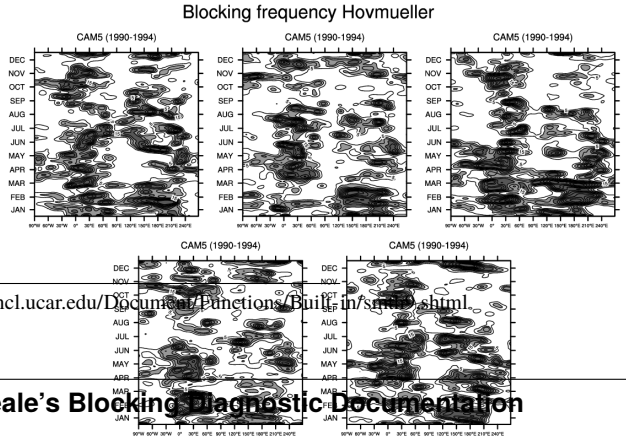
   - `case_names` (short names; for ensembles these are the group name repeated length ncase)
   - `file_names` (paths to files for each case (distinct for each ensemble member))
   - `var_names` (what the variables are named in the datasets)
   - `years_run0, years_run1` (start and stop years for each case)
   - `plot_colors` hard-coded for consistency from run to run

2. Sets up figures that need to be done before each season is processed *To be done*
3. Loops over seasons (tested on ANN. DJF & JJA not tested)

- Loops over files. File settings obtained by `file_opts = extract_file_opts(all_cases,ifile)`
  1. If `MDTF_BLOCKING_READ_DIGESTED = True`
     - Calls `blocking_digested.ncl:open_digested()` to open `file_opts@file_name` (set in `blocking_funcs.ncl:set_and_check_file_names()`) reads `block_time = f_dig->$var_name$(idays,:)`
     - `else` reads raw Z500 data using `var_in_p = blocking_readfile_orig` and computes the blocking index as `block_time(lon)` following *Tibaldi and Molteni (1990)* (page 51)
  2. If `MDTF_BLOCKING_WRITE_DIGESTED` Optionally writes `block_time` out as digested data
  3. computes `block_days` as sum of all days that were blocked over the entire time period, as a function of longitude (as well as std for ensembles) `block_std` and `block_freq` to be blocked/all days.
  4. for season `ANN` (annual) only, calculate daily frequency `block_freq_yr(ndoys,nlons)` for Hovmüllers, smoothed by smth9_Wrap[166]
- Loops over ensemble groups for figures
  1. Store data in arrays `block_freq_aves_p, block_freq_min_p/max_p`
  2. Set more plotting resources `res_m`
  3. Individual (Hovmueller figures) `plot_t(ifile) = gsn_csm_hov(wks_ens_page, lonPivot(block_freq_yr,270.), res_t)`
  4. Combined seasonal figure (one figure with a line for each dataset) `plot(ip) = gsn_csm_xy (wks,block_freq_aves_p&lon,block_freq_aves_p,res_m)`

Figures





3.2.4 References

1. Tibaldi and Molteni (1990): On the operational predictability of

[166] https://www.ncl.ucar.edu/Document/Functions/Built_in/smth9.shtml

blocking. *Tellus A: Dynamic Meteorology and Oceanography*, **42** (3), 343-365, doi:10.3402/tellusa.v42i3.11882[167].

´ 2. D'Andrea, F., Tibaldi, S., Blackburn, M. et al. (1998): Northern Hemisphere atmospheric blocking as simulated by 15 atmospheric general circulation models in the period 1979–1988. *Climate Dynamics*, **14**, 385–407 doi:10.1007/s003820050230[168].

´ 3. Davini, P., Cagnazzo, C., Neale, R., and Tribbia, J. (2012): Coupling between Greenland blocking and the North Atlantic Oscillation pattern, *Geophys. Res. Lett.*, **39**, L14701, doi:10.1029/2012GL052315[169].

3.3 Convective Transition Diagnostic Package

Last update: 12/2/2020

The convective transition diagnostic package computes statistics that relate precipitation to measures of tropospheric temperature and moisture, as an evaluation of the interaction of parameterized convective processes with the large-scale environment. Here the basic statistics include the conditional average and probability of precipitation, PDF of column water vapor (CWV) for all events and precipitating events, evaluated over tropical oceans. The critical values at which the conditionally averaged precipitation sharply increases as CWV exceeds the critical threshold are also computed (provided the model exhibits such an increase).

3.3.1 Version & Contact info

- Version 2 02-Dec-2020 Yi-Hung Kuo (UCLA)
- PI: J. David Neelin (UCLA; neelin@atmos.ucla.edu)
- Current developer: Yi-Hung Kuo (yhkuo@atmos.ucla.edu)
- Contributors: K. A. Schiro (UCLA), B. Langenbrunner (UCLA), F. Ahmed (UCLA), C. Martinez (UCLA), and C.-C. (Jack) Chen (NCAR)

[167] https://doi.org/10.3402/tellusa.v42i3.11882
[168] https://doi.org/10.1007/s003820050230
[169] https://doi:10.1029/2012GL052315

Open source copyright agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.3.2 Functionality

The currently package consists of following functionalities:

1. Convective Transition Basic Statistics (convecTransBasic.py)
2. Convective Transition Critical Collapse (convecTransCriticalCollape.py)
3. (*) Precipitation Contribution Function (cwvPrecipContrib.py)

More on the way... (* under development)

As a module of the MDTF code package, all scripts of this package can be found under the convective_transition_diag[170] directory and digested observational data under `inputdata/obs_data/convective_transition_diag`.

3.3.3 Required programming language and libraries

The is POD is written in Python 3.7, and requires the following Python packages: os, glob, json, Dataset, numpy, scipy, matplotlib, networkx, warnings, numba, & netcdf4. These dependencies are included in the python3_base environment provided by the automated installation script for the MDTF Framework. Note that running this POD outside the provided environment may result in figures different from the samples[171].

3.3.4 Required model output variables

The following three 3-D (lat-lon-time) high-frequency model fields are required:

1. Precipitation rate (units: mm s^{-1} = kg m^{-2} s^{-1}; **6-hrly avg. or shorter**)
2. Column water vapor (CWV, or precipitable water vapor; units: mm = kg m^{-2})
3. Column-integrated saturation humidity (units: mm = kg m^{-2}) or mass-weighted column average temperature (units: K), column: 1000-200 hPa by default. Since variables in (3) are not standard model output, this package will automatically calculate (3) if the following 4-D (lat-lon-pressure-time) model field is available:
4. Air temperature (units: K)

3.3.5 References

1. Kuo, Y.-H., J. D. Neelin, and C. R. Mechoso, 2017: Tropical Convective Transition Statistics and Causality in the Water Vapor-Precipitation Relation. *J. Atmos. Sci.*, **74**, 915-931, https://doi.org/10.1175/JAS-D-16-0182.1.
2. Kuo, Y.-H., K. A. Schiro, and J. D. Neelin, 2018: Convective transition statistics over tropical oceans for climate model diagnostics: Observational baseline. *J. Atmos. Sci.*, **75**, 1553-1570, https://doi.org/10.1175/JAS-D-17-0287.1.
3. Kuo, Y.-H., and Co-authors, 2020: Convective Transition Statistics over Tropical Oceans for Climate Model Diagnostics: GCM Evaluation. *J. Atmos. Sci.*, **77**, 379-403, https://doi.org/10.1175/JAS-D-19-0132.1.

[170] https://github.com/NOAA-GFDL/MDTF-diagnostics/tree/main/diagnostics/convective_transition_diag

[171] http://www.cgd.ucar.edu/cms/bundy/Projects/diagnostics/mdtf/mdtf_figures/MDTF_QBOi.EXP1.AMIP.001.save/convective_transition_diag/convective_transition_diag.html

See http://research.atmos.ucla.edu/csi//REF/pub.html for updates.

3.3.6 More about this diagnostic

The current version of the convective transition diagnostic package produces three sets of figures for both pre-digested observations and model output, including (1) basic statistics, (2) collapsed statistics, and (3) critical column water vapor. In the following, we will show an example set of the figures for an uncoupled simulation of the 1° version of the GFDL AM4 (configuration AM4-G9; *Zhao et al., 2018a* (page 119), *2018b* (page 56); see also *Kuo et al., in prep* (page 119)) that are produced by the package.

1) Basic statistics

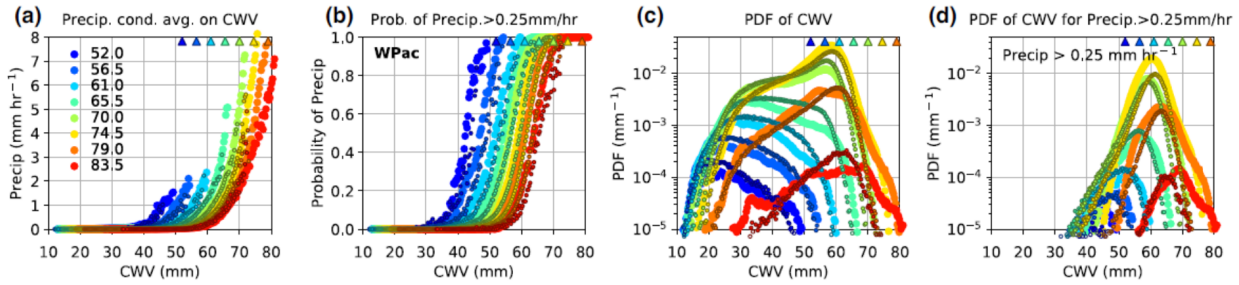


Fig. 2: Basic statistics computed using events over tropical western Pacific (20°S-20°N, west to 180°), including (a) conditionally averaged precipitation rate, (b) conditional probability of precipitation > 0.25 mm hr^{-1}, (c) PDF of CWV, and (d) PDF of CWV for precipitating events, all as a function of CWV. Here the large markers represent results simulated by the model, and small markers represent the corresponding observations at 1°. The colors indicate the column-integrated saturation humidity which is used as a bulk measure of tropospheric temperature (also shown as triangles). The PDFs in (c) together represent the normalized joint PDF of CWV and bulk temperature. Multiplying (b) and (c) results in PDFs in (d) (not normalized).

The observed (small markers) and simulated precipitation (large markers) in panel (a) sharply picks up as CWV exceeds a certain threshold, known as the critical CWV [see panel (e) below for how it is defined, and panel (i) for the values]. Here, the column integrated saturation humidity $\widehat{q_{sat}}$ (units: mm) is used as a bulk measure of the tropospheric temperature. As the bulk tropospheric temperature increases, the pickup of precipitation occurs at higher CWV. The probability of precipitation in panel (b) exhibits a similar pickup behavior. The AM4 model examined here can reasonably simulate the observed pickup of precipitation, with slightly higher probability than observed.

In panel (c), the observed PDFs of CWV display characteristic shapes that depend on the bulk tropospheric temperature. At low temperature, the PDF peaks at a low CWV value, below which the PDF drops rapidly, and above which the PDF decreases slowly until reaching a cutoff . As temperature increases, another peak around critical develops with the low-CWV peak diminishing. The rapid drop of PDF for CWV above critical [see panel (g) below] is consistent with the pickup of precipitation, i.e., precipitation becomes an effective moisture sink in this regime. It has been noted that low-level convergence tends to be associated with high-CWV events, while low-level divergence is associated with low-CWV events. The AM4 model reasonably reproduces the observed CWV PDF with noticeably more above-critical events. However, given the uncertainty associated with the CWV retrievals used here (RSS TMI data products, version 7.1; *Wentz et al. 2015* (page 119)), especially at high values, we cannot conclude that the model misbehaves in the high-CWV regime.

2) Collapsed statistics

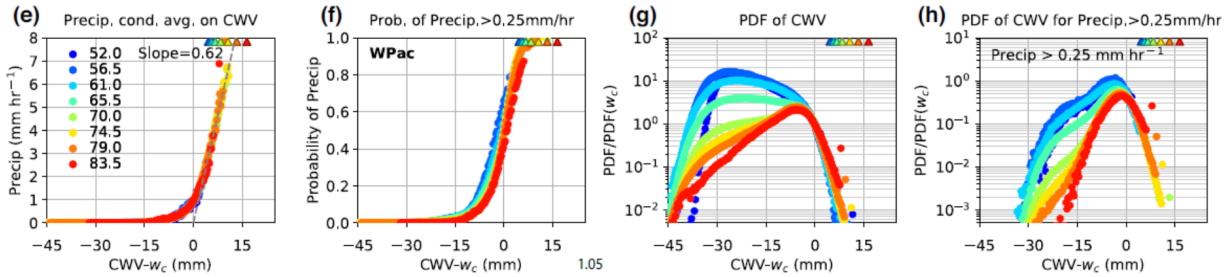


Fig. 3: Same as the statistics in panels (a)-(d), respectively, but for each bulk tropospheric temperature, shift the CWV by the corresponding critical CWV w_c. Here, only the results from observations are presented. In panels (g)-(h), the PDF values are rescaled.

In practice, we define the critical CWV to be value at which the best-fit line to the conditionally averaged precipitation intersects with the CWV axis, as shown in panel (e) [see panels (i)-(j) below for the observed and simulated critical values]. When expressed as a function of CWV w_c, the conditional average and probability of precipitation [panels (e)-(f)] collapse without exhibiting dependence on the bulk temperature (and ocean basin). The rescaled PDFs in panel (g) also collapse for CWV above critical. For the most relevant temperature bins in the tropics ($\widehat{q_{sat}}$ 70 mm or the mass-weighted column average temperature 271 K), the PDF of CWV for precipitating events share a common near-Gaussian core near the critical CWV.

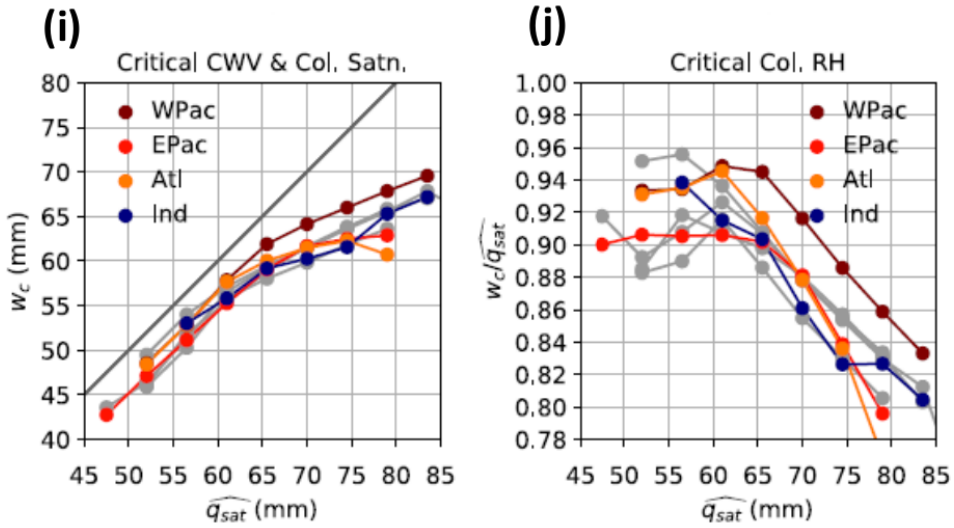


Fig. 4: (i) Critical CWV w_c and (j) the corresponding critical column relative humidity $w_c/\widehat{q_{sat}}$. The colored markers represent the results simulated by the AM4 model and observed values in gray.

Compared to the observations, the slope of the best-fit line simulated by the AM4 model is slightly higher than observed [0.76 vs. 0.62 in panel (e); *Kuo et al., in prep* (page 119)], but within the uncertainty range of observations (*Kuo et al. 2018* (page 119)). The simulated statistics are more sensitive to the tropospheric temperature and ocean basin and indicate that there are more above critical events for highest temperature bins in the model. The functional form of the PDFs for precipitating events deviates from Gaussian. The simulated critical values are consistent with the observed values [panels (i)-(j)]. It has been noted that the dependence of critical values on tropospheric temperature (i.e., critical CWV increases with tropospheric temperature but the corresponding critical column RH $w_c/\widehat{q_{sat}}$ decreases) is a generic consequence of including entrainment in the buoyancy/conditional instability calculation (*Sahany et al. 2012* (page 119)).

3.3.7 Additional references

4. Sahany, S., J. D. Neelin, K. Hales, and R. B. Neale, 2012: Temperature–moisture dependence of the deep convective transition as a constraint on entrainment in climate models. *J. Atmos. Sci.*, **69**, 1340–1358, https://doi.org/10.1175/JAS-D-11-0164.1.

5. Wentz, F.J., C. Gentemann, K.A. Hilburn, 2015: Remote Sensing Systems TRMM TMI Daily, 3-Day Environmental Suite on 0.25 deg grid, Version 7.1. Remote Sensing Systems, Santa Rosa, CA. Available online at https://www.remss.com/missions/tmi.

6. Zhao., M., and Coauthors, 2018a: The GFDL Global Atmosphere and Land Model AM4.0/LM4.0 - Part I: Simulation Characteristics with Prescribed SSTs. *Journal of Advances in Modeling Earth Systems*, **10(3)**, https://doi.org/10.1002/2017MS001208.

7. Zhao., M., and Coauthors, 2018b: The GFDL Global Atmosphere and Land Model AM4.0/LM4.0 - Part II: Model Description, Sensitivity Studies, and Tuning Strategies. *Journal of Advances in Modeling Earth Systems*, **10(3)**, https://doi.org/10.1002/2017MS001209.

3.4 Moist Static Energy Diagnostic Package

Last update 01/21/2021

ENSO moist static energy (MSE) diagnostic package consists of four levels. With a focus on identifying leading processes that determine ENSO-related precipitation anomalies, main module of the process-oriented diagnostic (POD) estimates vertically MSE budget and its variance analysis to account for relative contribution of each MSE term to column MSE. In that pursuit, POD is applied to monthly data (climate model or reanalysis products), and budget terms are estimated for "composite" El Niño or La Nina events. To estimate MSE budget, along with surface and radiation fluxes, 3-dimensional atmospheric variables are required. Hence, ERA-Interim is "considered" as "observations" here, and diagnostics obtained from ERA-Interim are used for model validation. In this general document, brief descriptions of the four levels of the POD are provided but detailed information (e.g., equations and input variables) is provided at each level. For the four levels of diagnostics, selected results are illustrated here.

POD works efficiently if model data contain sufficient number of El Niño or La Nina events.

To obtain robust results, a reasonable number of El Nino events is preferred in any model simulation (AMIP or CMIP) as predigested results correspond to results obtained from CMIP historical simulations for the period 1951-2005 in which 7-9 events are identified. Since the leading MSE terms are invariant from one event to another, useful information can still be obtained from shorter model runs in which 1-2 events are only identified.

Version and contact information

Version 1, revision (second) 01/21/2021

PI: Dr. H. Annamalai (IPRC/SOEST University of/ Hawaii; hanna@hawaii.edu)

Current developer: Jan Hafner (IPRC/SOEST University of Hawaii; jhafner@hawaii.edu)

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

Functionality

The current package consists of the following functionalities:

1. Basic ENSO diagnostics performed by script COMPOSITE.py
2. MSE (Moist Static Energy) budget analysis performed by script MSE.py

3. MSE variance diagnostics performed by script MSE_VAR.py
4. MSE scatter plots performed by script SCATTER.py

As a module of the MDTF code package, all scripts to perform the four levels can be found here: ~/diagnostics/ENSO_MSE/

The pre-digested observational data for model validation can be found here: ~/diagnostics/inputdata/obs_data/ENSO_MSE/

Required programming language and libraries

This package is coded in Python 3.8.5 and requires the following packages: numpy, os, math, xarray, netcdf4.

The pre-processing and plotting are coded in NCAR Command Language Version 6.5.0.

Required model output variables and their corresponding units

The following model fields are required as monthly data:

4-D variables (longitude, latitude, pressure level, time):

1. *zg* : HGT geopotential height (m)
2. *ua* : U wind component [m/s]
3. *va* : V wind component [m/s]
4. *ta* : Temperature [K]
5. *hus* : Specific Humidity [kg/kg]
6. *wap* : Vertical Velocity [Pa/s]

3-D variables (longitude, latitude, time):

7. *pr* : Precipitation [kg/m2/s]
8. *ts* : Surface Temperature [K]
9. *hfss* : Sensible Heat Flux [W/m2]
10. *hfls* : Latent Heat Flux [W/m2]
11. Net Shortwave Radiative Flux [W/m2]
12. Net Longwave Radiative Flux [W/m2]

Net radiative fluxes [variables 11 and 12] are estimated from the individual seven (7) radiative flux components, namely: (i) *rsdt* : Top Of Atmosphere (TOA) Shortwave down; (ii) *rsut* : TOA Shortwave up; (iii) *rsds* : Surface Shortwave down; (iv) *rsus* : Surface Shortwave up; (v) *rlut* : TOA Longwave up (OLR); (vi) *rlus* : Surface Longwave up and (vii) *rlds* : Surface Longwave down.

More details can be found in the README_general.pdf document.

References

1. Annamalai, H., 2020: ENSO precipitation anomalies along the equatorial Pacific: Moist static energy framework diagnostics. *J. Climate*, **33**, 9103-9127, doi:10.1175/JCLI-D-19.0374.1 https://doi.org/10.1175/JCLI-D-19-0374.1

2. Annamalai, H., J. Hafner, A. Kumar and H. Wang, 2014: A framework for dynamical seasonal prediction of precipitation over Pacific islands. *J. Climate*, **27**, 3272-3297, https://doi.org/10.1175/JCLI-D-13-00379.1

More about this diagnostic

Level 1 – Basic ENSO diagnostics

Composites, regression and correlation etc: Reference index (e.g., Nino3.4 SST)

- Monthly and seasonal averages
- 2 Year life cycle of ENSO: Year (0) and Year (1)

Year (0) = developing phase and Year (1) = decaying phase

At this level, POD calculates simple seasonal averages, composites, regression and correlations. Based on a reference ENSO index (e.g., area-averaged SST anomalies over Nino3.4 region), seasonal composites of variables relevant to MSE budget are constructed for the entire 2-year life cycle of ENSO. Here, Y (0) refers to the developing, and Y (1) the decaying phase of ENSO. Fig. 1 shows composite net radiative flux divergence in the column (Frad) for boreal winter (DJF) season during El Niño (Fig. 1a) and La Nina (Fig. 1b) constructed from CCSM4 historical simulations.

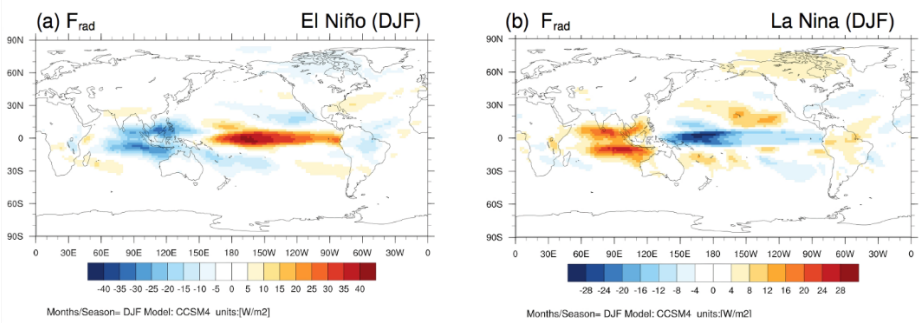


Figure 1: Boreal winter (DJF) composites of net radiative flux divergence in the column (Frad) constructed from CCSM4 historical simulations (1950-2005): (a) El Niño and (b) La Nina

More details on Level 1 diagnostics can be found in the README_LEVEL_01.pdf document.

Level 2 – MSE (Moist Static Energy) budget analysis (for composite ENSO)

In the deep tropics, weak temperature gradient approximation (WTG) implies that precipitation is largely determined by column MSE [or vertically integrated specific humidity or precipitable water (PW)]. Fig. 2a shows that in regions of organized positive and negative precipitation anomalies along the equatorial Pacific, spatial structure and intensity of MSE (contour) and PW (shading) anomalies bear a "close association". In this view, climate model biases in column MSE and precipitation are clearly linked and models' fidelity in representing ENSO-related precipitation anomalies along the equatorial Pacific then requires that models accurately represent processes that determine column MSE anomalies.

In Level 2, for the composites constructed in Level 1, vertically integrated MSE and its budget are estimated (more details on the equations etc., are in the README file in Level 2). All the terms are expressed in energy units (W/m2). As an illustration, anomalous MSE export (or vertical advection of MSE) calculated for composite El Niño winter from CCSM4 solutions is shown in Fig. 2b.

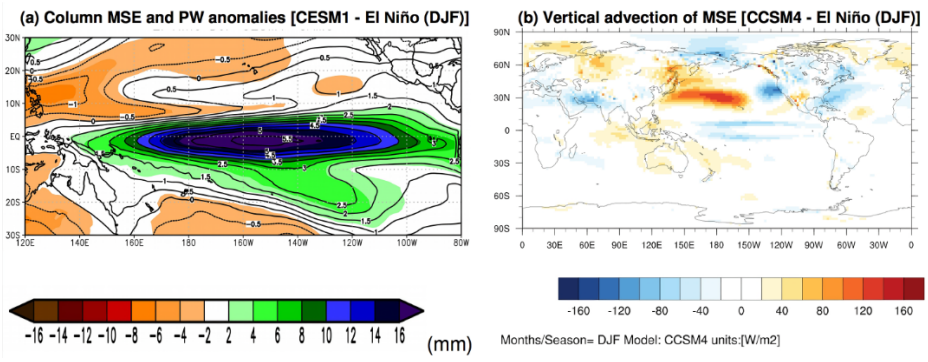


Figure 2: (a) Vertically integrated anomalous MSE (contours, J/m2, and scaled by 10E-7) and precipitable water (shaded, mm) and (b) vertical advection of MSE (W/m2). Results are for composite El Niño winters.

More details on Level 2 diagnostics can be found in the README_LEVEL_02.pdf document.

Level 3 – MSE variance diagnostics (for composite ENSO)

Once all the individual MSE terms are estimated their relative contributions to column MSE is estimated here. This particular diagnostic is estimated for user-defined regions of interest, and outputs correspond to co-variances scaled by MSE variance (equation details in Level 3 README file). For example, one can estimate this diagnostic for equatorial central and eastern Pacific regions separately to assess the role of different processes in contributing to column MSE anomalies (or precipitation anomalies).

In the current version, the diagnostic is estimated for two "default regions" and one user optional region, and they are:

a): Equatorial Central Pacific 180°–200°E 10°S – 5°N

b): Equatorial Eastern Pacific 220°–280°E 5°S – 5°N

c): User prescribed area (for more details see README_LEVEL_03 document)

Here, in Fig. 3 results for both composite El Niño and La Nina winters, and from both equatorial central and eastern Pacific regions obtained from CCSM4 solutions are shown.

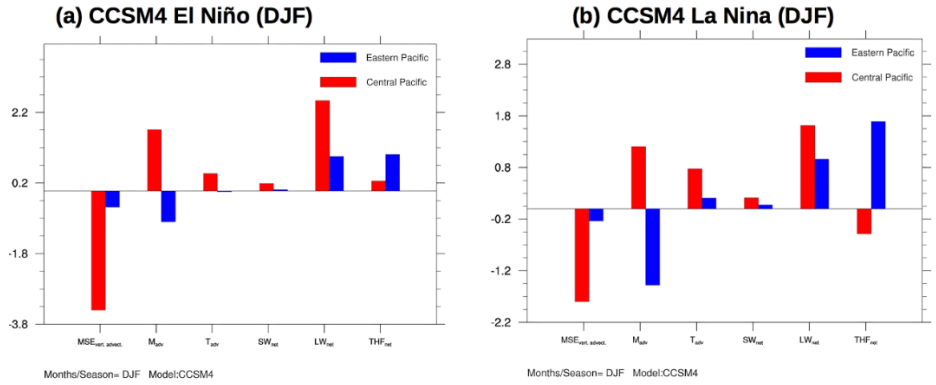


Figure 3: Relative contributions of various MSE terms to column MSE averaged for equatorial central (red) and eastern (blue) Pacific regions estimated from CCSM4 historical solutions for composite: (a) El Niño winter and (b) La Nina winter.

More details on Level 3 diagnostics can be found in the README_LEVEL_03.pdf document.

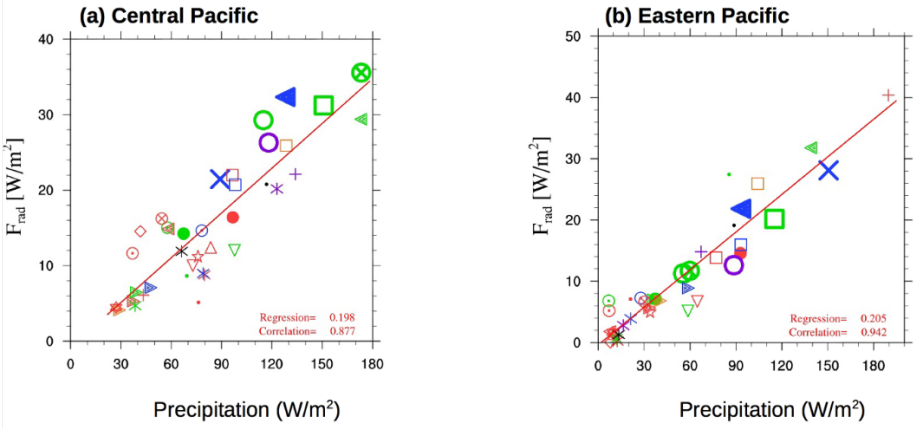
Level 4 – MSE scatter plots (Metrics).

Note that if diagnostics from multiple models are sought to assess systematic errors across all models then the results can be displayed as scatter plots between variables that are physically linked. In this level, one can also estimate "inter-model correlations" and "best fit" regression line, and show them in the plots.

At this level, results from Level 2 (CMIP-era models) are condensed into scatter plots. Specifically, estimates of each MSE budget term (e.g., Frad) is plotted against precipitation, and the example shown here in Fig. 4 suggests that error in representing net radiative flux divergence (Frad) is systematically tied to error in model simulated precipitation over both the equatorial central and eastern Pacific regions.

Figure 4. Scatter plots between anomalous net radiative flux divergence (Frad) and precipitation for composite El Niño winters estimated from historical simulations of CMIP5 models: (a) Central Pacific and (b) Eastern Pacific. In the panels, inter-model correlations and best regression fit lines are also provided.

More details on Level 4 diagnostics can be found in the README_LEVEL_04.pdf document.



3.5 Rossby Wave Sources Diagnostic Package

Last update 04/15/2021

ENSO Rossby wave sources (ENSO_RWS) diagnostic package consists of four levels. With a focus on identifying leading processes that determine ENSO-induced global teleconnection, particularly the Pacific North American (PNA) pattern, the main module of the POD estimates basic state flow properties at an appropriate tropospheric upper-level and solves the barotropic vorticity equation to estimate various terms that contribute to the total anomalous RWS. In that pursuit, the ENSO-RWS POD is applied to monthly data (climate model or reanalysis products), and RWS terms are estimated for "composite" El Niño or La Nina events. To attain robust "composite" results a reasonable sample of ENSO winters is needed. However, the POD can be applied even for a single El Niño winter (e.g., when applied to seasonal prediction models). Similarly, the POD is applicable to any number of pressure levels (e.g., to identify the level at which maximum upper-level divergence and associated RWS are located). Here, reanalysis products (e.g., ERA-Interim) are considered as observations and diagnostics obtained from ERA-Interim and other reanalysis products are used for model validation. In this general document, brief descriptions of the four levels of the POD are provided, and detailed information is provided at each level. For the four levels of diagnostics, selected results are illustrated here.

The POD works efficiently if the model data contain a sufficient number of El Niño or La Nina events. Predigested results are available for both El Niño and La Nina composites.

Version and contact information

Version 1, 03/09/2021

PI: Dr. H. Annamalai (IPRC/SOEST University of/ Hawaii; hanna@hawaii.edu)

Current developer: Jan Hafner (IPRC/SOEST University of Hawaii; jhafner@hawaii.edu)

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

Functionality

The current package consists of the following functionalities:

1. Basic ENSO diagnostics performed by script LEVEL_01.py
2. Climatological flow properties script LEVEL_02.py
3. Rossby wave source terms performed by script LEVEL_03.py
4. Scatter plots as metrics to assess models performed by script LEVEL_04.py

As a module of the MDTF code package, all scripts to perform the four levels can be found here: ~/diagnostics/ENSO_RWS/

The pre-digested observational data for model validation can be found here: ~/diagnostics/inputdata/obs_data/ENSO_RWS/

Required programming language and libraries

This package is coded in Python 3.8.5 and requires the following packages: numpy, os, math, xarray, netcdf4.

The pre-processing and plotting are coded in NCAR Command Language Version 6.5.0.

Required model output variables and their corresponding units

The following model fields are required as monthly data:

4-D variables (longitude, latitude, pressure level, time):

1. *zg*: HGT geopotential height (m)
2. *ua*: U wind component [m/s]
3. *va*: V wind component [m/s]
4. *ta*: Temperature (K)
5. *wap*: Vertical velocity (Pa/s)

3-D variables (longitude, latitude, time):

6. *pr*: Precipitation (k/m^2/s)
7. *ts*: Surface temperature (K)

More details can be found in the README_general.pdf document.

References

1. Annamalai, H., R. Neale and J. Hafner: ENSO-induced teleconnection: development of PODs to assess Rossby wave sources in climate models (in preparation).

2. R. Neale and H. Annamalai: Rossby wave sources and ENSO- induced teleconnections in CAM6 model development versions and associated vertical processes (in preparation).

More about ENSO_RWS

Rossby Wave Source POD to assess ENSO-induced teleconnection (ENSO_RWS)

The NOAA-MDTF Rossby Wave Source (RWS) Process Oriented Diagnostic (POD) package fills a critical gap in the diagnostics tools available to climate model developers. In both basic-state and anomalous conditions, changes in the response of moist processes in model either parameterization modifications or tuning and calibration can often change the nature of the seasonal distributions of tropical precipitation and associated heating, and by association moistening and divergence profiles. While validation of precipitation is straightforward, an understanding of the circulation consequences in the tropics and the extra-tropics is not.

The RWS POD developed here will help to address this critical validation gap by quantifying the roles of changing ambient flow properties (basic state climatological features), and anomalous upper tropospheric divergent patterns in the generation and radiation of planetary stationary Rossby waves. This will be particularly important in coupled configuration development, since changes to the atmosphere model configuration can lead to complex coupled feedback modifying not only the mean climate, but the response during ENSO events.

A further role this POD plays is in determining the potential for generated interactions to influence the United States, and whether a particular model version has prediction utility.

Few take home messages to model development include:

- Ambient flow properties (e.g., restoring force for stationary Rossby waves)

- Perturbations to local Hadley and planetary east-west Walker circulations (generation of Rossby wave sources)
- Location and intensity of anomalous Rossby wave sources and their dependence on both ambient and anomalous circulation conditions (principal Rossby wave sources)
- Radiation of Rossby waves in the presence of ambient flow properties (great circle path)
- To infer ENSO-induced seasonal anomalies over North America (prediction utility)

Level 1 – Basic ENSO diagnostics

Identify ENSO winters and construct seasonal composite anomalies for relevant variables (e.g., anomalous precipitation, circulation, geopotential height to estimate standardized PNA index).

Reference index (e.g., Nino3.4 SST)

- Seasonal averages

Based on a reference ENSO index (e.g., area-averaged SST anomalies over the Nino3.4 region), seasonal composites of variables relevant to ENSO-induced global teleconnection at an appropriate tropospheric upper-level are constructed. Fig. 1 shows composite anomalous precipitation (shaded), 200hPa divergence/convergence (contour/hatching) and 200hPa divergent wind (vector) for boreal winter (DJF) season during El Niño constructed from GFDL-CAM4 AMIP simulations performed for the period 1980-2014 (Fig. 1a) and ERA-interim (Fig. 1b).

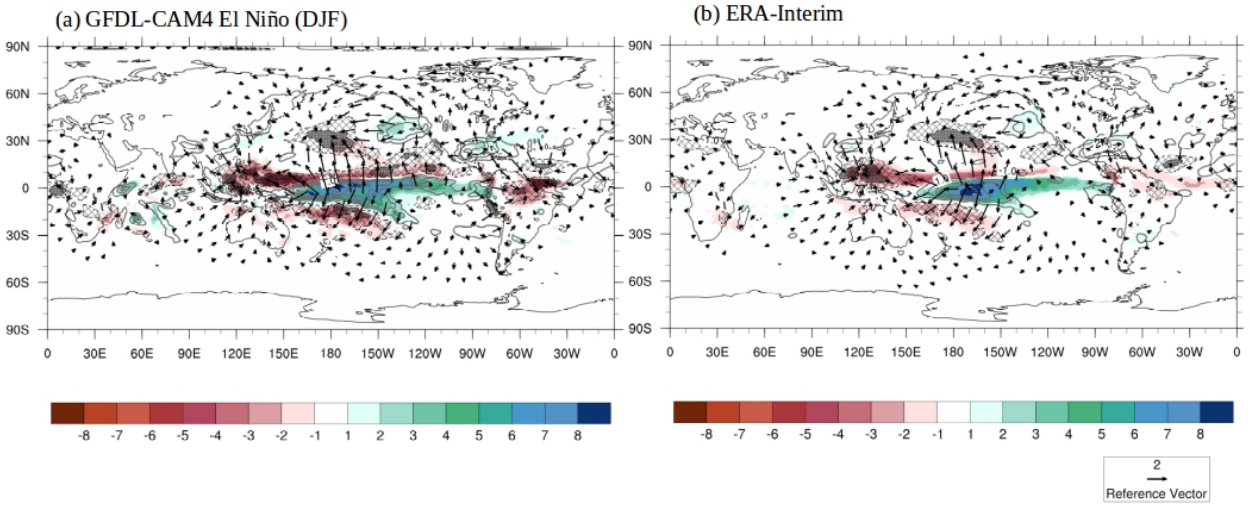


Figure 1: El Niño winter (DJF) composites of precipitation anomalies (shaded; mm/day), anomalous 200hPa convergence/divergence (contours/hatching in units of 10^{-6} s^{-1}) and anomalous 200hPa divergent wind anomalies (m/s) constructed from: (a) AMIP simulation of GFDL-AM4 performed for the period 1980-2014 and (b) ERA-interim. Reference wind vector is also shown.

More details on Level 1 diagnostics can be found in the README_LEVEL_01.pdf document located in ~/diagnostics/ENSO_RWS/doc.

Level 2 – Climatological flow and wave properties (basic-state/ambient flow) diagnostics

Regarding to basic or climatological flow properties, restoring effect for Rossby waves ($*$) that is dependent on meridional gradient in absolute vorticity () and meridional curvature of the zonal flow or gradients in relative vorticity $\frac{\partial^2 U}{\partial y^2}$ and resultant stationary wave number (K_s) are diagnosed. These ambient flow properties determine generation and propagation of stationary Rossby waves.

Mathematical expressions for $*$ and K_s are given by:

$\beta_* = \beta - \frac{\partial^2 U}{\partial y^2}$ (1)

$K_s = \left(\beta_*/U\right)^{1/2}$ (2)

where is latitudinal variations in planetary vorticity (f), $\acute{U}$ is the basic-state zonal wind velocity, and $\frac{\partial^2 U}{\partial y^2}$ is the curvature of the ambient zonal flow. Stationary Rossby waves are possible if the flow is westerly ($\acute{U}$ positive) and β_* is positive.

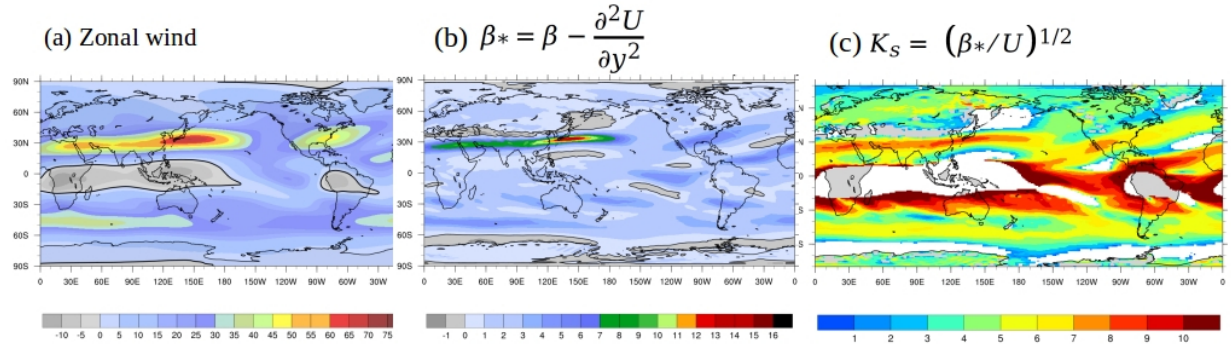


Figure 2: GFDL-AM4 simulated ambient flow properties at 200hPa for boreal winter (December – February): (a) zonal wind (m/s); (b) $\beta(10^{-11}\text{m}^{-1}\text{s}^{-1})$ and (c) stationary wavenumber. In (a and b), negative values are shaded gray and zero contour is shown as thick line. In (c) unspecified or singular values of wavenumber is shown as white.

More details on Level 2 diagnostics can be found in the README_LEVEL_02.pdf document located in ~/diagnostics/ENSO_RWS/doc.

Level 3 – Rossby wave sources (for composite ENSO)

Explicitly solves barotropic vorticity budget and the leading terms contributing to the total anomalous Rossby wave sources (RW$S^{'}$) are quantified. The mathematical expression for RW$S^{'}$ is given by:

$$\text{RWS}^{'} = -\bar{\xi}\nabla\bullet v_{\chi}^{'} - v_{\chi}^{'}\bullet\nabla\bar{\xi} - \xi^{'}\nabla\bullet\bar{v}_{\chi} - \bar{v}_{\chi}\bullet\nabla\xi^{'} \quad (3)$$

Here, ξ and v_χ correspond to absolute vorticity and divergent component of the wind, respectively. The overbar represents seasonal mean and the prime refers to seasonal anomalies. The first term in RW$S^{'}$ corresponds to stretching due to anomalous divergence, and the second term accounts for advection of climatological gradient in ξ by the anomalous divergent wind. The third and fourth terms account for transient eddy convergence of vorticity, and their contributions to RW$S^{'}$ is small but non-negligible.

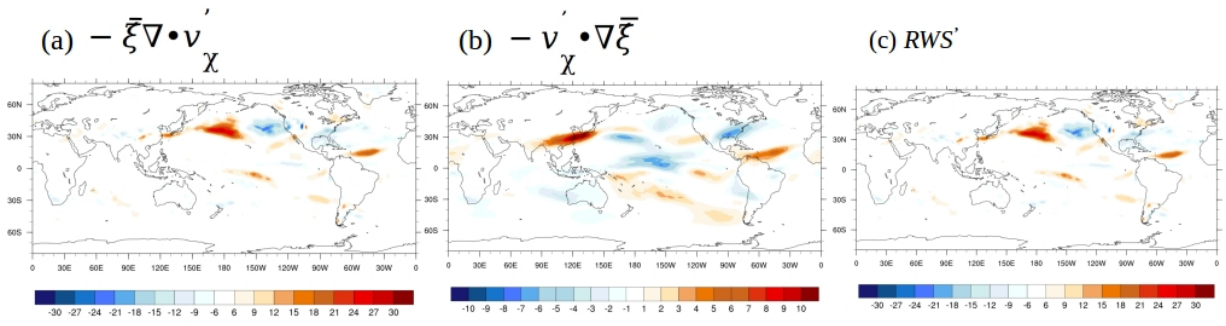


Figure 3: Anomalous Rossby wave sources (10^{-11}s^{-2}) due to: (a) stretching term; (b) anomalous divergent wind advecting gradient in climatological absolute vorticity and (c) all the four terms (equation 3). Results shown are for composite El Niño winters (DJF) simulated by GFDL-AM4 AMIP simulations.

More details on Level 3 diagnostics can be found in the README_LEVEL_03.pdf document located in ~/diagnostics/ENSO_RWS/doc.

Level 4 – Scatter plots for assessing models' performance (Metrics).

Note that if diagnostics from multiple models are sought to assess systematic errors across all models and/or compare and contrast a selected model's performance with other models then the results can be displayed as scatter plots between variables that are physically linked. At this level, results from Levels 1-3 are condensed into scatter plots. Specifically, estimates of leading anomalous RWS terms are plotted against equatorial precipitation and/or standardized PNA index (defined from 200hPa height anomalies).

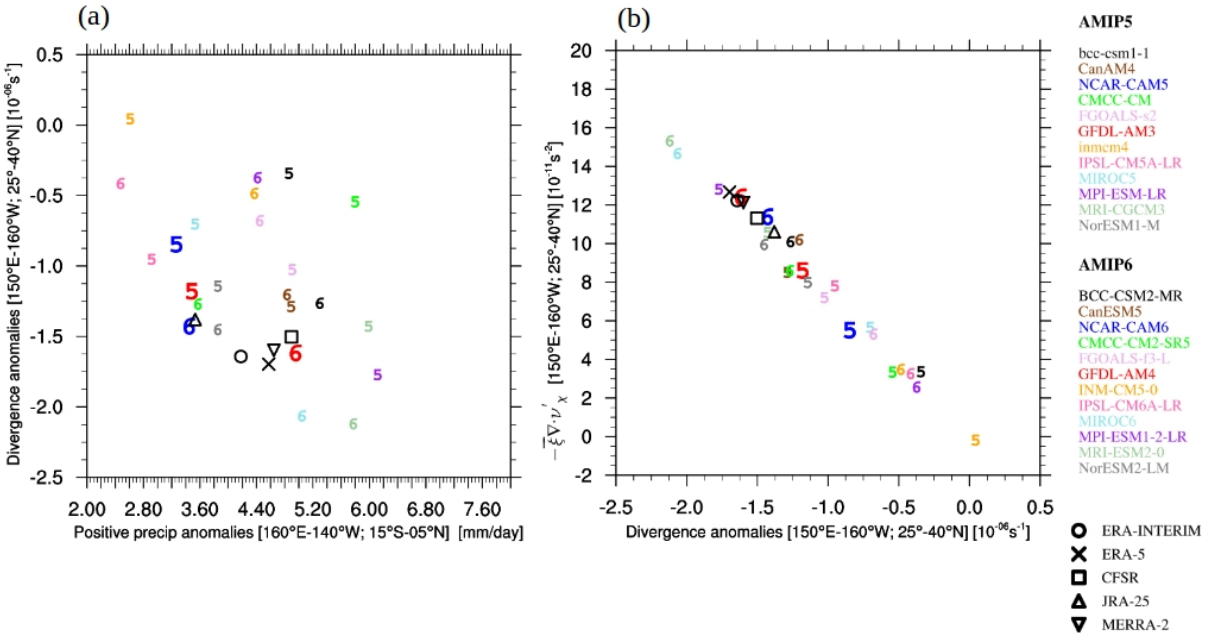


Figure 4: Scatter plots between (a) anomalous equatorial Pacific precipitation (160°E-140°W; 15°S-0) and 200hPa divergence (150°E-160°W; 25°N-40°N); (b) anomalous 200hPa divergence and RWSdue to stretching term (150°E-160°W; 25°N-40°N); (c) anomalous total RWS' east of the dateline (180°E-160°W; 25°-40°N) and standardized PNA index and (d) 200hPa climatological βzero value longitude with respect to dateline and standardized PNA index. Results shown are for composite El Niño winters (DJF) simulated by AMIP5/6 models. In the panels, number 5 corresponds to AMIP5 and 6 corresponds to AMIP6 models, and the color of the numbers correspond to the model's name.

More details on Level 4 diagnostics can be found in the README_LEVEL_04.pdf document located in ~/diagnostics/ENSO_RWS/doc.

3.6 EOF of Geopotential Height Diagnostic Module From NCAR

Last update: 03/11/2019

3.6.1 Contact info

- Current Developer: Dani Coleman (bundy@ucar.edu), NCAR
- Contributors: Dennis Shea, Andrew Gettleman, Jack Chen (NCAR)

This computes the climatological anomalies of 500 hPa geopotential height, then calculates the EOFs using NCL's eofunc. The code is in NCL and requires model input:

1. monthly averaged surface pressure (ps),
2. monthly averaged geopotential height (zg).

- Generates a netcdf file of climatological anomalies of 500 hPa geopotential height (compute_anomalies.ncl)
- Calculates and plot EOFs of North Atlantic (eof_natlantic.ncl) and North Pacific regions using NCL function eofunc
- Uses pre-made figures of eofs of NCEP observational data for comparison.

3.6.2 Open source copyright agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.6.3 Functionality

All scripts can be found at: `mdtf/MDTF_$ver/var_code/EOF_500hPa`

1. Make anomalies (compute_anomalies.ncl)

2. Calculated and plots EOFs in N. Atlantic (eof_natlantic.ncl) and N. Pacific (eof_npacific.ncl)

Preprocessed observational data from NCEP as gif images are located in `mdtf/``inputdata/obs_data/EOF_500hPa`

Place your input data at: `inputdata/model/$model_name/day` index.html can be found at: `mdtf/MDTF_$ver/wkdir/MDTF_$model_name`

3.6.4 Required Programing Language and libraries

All these scripts required NCAR Command Language Version 6.3.0 or higher

Required input data to the module:

1) Monthly averaged surface pressure (ps)

2) Monthly averaged geopotential height (zg)

3.7 Eulerian Storm Track

Jeyavinoth Jeyaratnam[1] and James F. Booth[1]

[1]The City College of New York, CUNY, New York

Last update: 10/9/2020

3.7.1 Description

Synoptic variability in the atmosphere can be isolated by filtering atmospheric data temporally, in a manner that removes the diurnal and the greater than weekly variability (*Blackmon et al. 1976* (page 67)). Then, the standard deviation of the filtered data at each latitude and longitude can be interpreted as the climatological baroclinic wave activity, which, for historical reasons, is termed the storm tracks (*Wallace et al. 1988* (page 67)). Because these storm tracks are calculated for each latitude- longitude point using time-series data, rather than tracking individual storms, they are sometimes referred to as the Eulerian storms tracks – as opposed to the Lagrangian storm tracks. The storm tracks are a large-scale metric for the skill in the model representation of baroclinic wave behavior – which includes extratropical cyclones. Storm track location, seasonality and intensity correlate very strongly with transient poleward energy transport (*Chang et al. 2002* (page 67)).

Storm tracks can be evaluated with atmospheric data such as meridional wind or geopotential height (see *Chang et al. 2002* (page 67) for a comparison of many different fields). *Booth et al. (2017)* (page 67) show that storm track strength – defined as the area-average of the storm track over an ocean basin, using meridional winds at 850 hPa correlate very strongly with the storm track at 500 hPa. This is true for interannual variability and for a comparison across multiple models. Therefore, the metric in this diagnostic calculates the storm track using meridional winds at 850 hPa. The nomenclature and calculation follow that of *Booth et al. (2017)* (page 67).

To isolate the synoptic timescale, this algorithm uses 24-hour differences of daily-averaged data. Using daily averages removes the diurnal cycle and the 24-hour differencing removes variability beyond 5 days (*Wallace et al. 1988* (page 67)). After filtering the data to create anomalies, the variance of the anomalies is calculated across the four seasons for each year. Then the seasonal variances are averaged across all years. For the first year in the sequence, the variance for JF is calculated and treated as the first DJF instance. For the final December in the sequence is not used in the calculation.

The maximum strength of the Eulerian storm track can be sensitive to the data's spatial resolution. To exemplify this fact, we have included the map view of the storm track using ERA- Interim and ERA5 reanalysis data at two different resolutions (1.5^{o} horizontal resolution for ERA- Interim data and 1^{o} resolution for ERA5 data). For this reason, we do not include a difference plot of the lat/lon storm track maps. Instead, for a side-by-side comparison, we have generated a zonal mean of the storm tracks.

3.7.2 Version & Contact Information

- Version 1.0 :: 10/09/2020
- Current Developer: Jeyavinoth Jeyaratnam (jjeyaratnam@ccny.cuny.edu), CUNY
- PI: James F. Booth (jfbooth@ccny.cuny.edu), CUNY

Open source copyright agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.7.3 Functionality

The POD contains the following files which perform various functionalities:

1. Eulerian_storm_track.py is the main driver code.
2. Eulerian_storm_track util.py is the code that computes the statistics.
3. plotter.py is the code used to create the plots.
4. Eulerian_storm_track_obs.py is an internal code used to preprocess the observations and convert them to NetCDF files.

3.7.4 Required programming language and libraries

This package is run using Python 3, and requires the following Python packages:

- os
- numpy
- xarray
- netCDF4
- matplotlib
- cartopy
- basemap

3.7.5 Required model output variables

The following 3D (time, lat, lon) model fields are required:

- V850 (units: m/s, daily)

3.7.6 References

1. Blackmon, M.L., 1976: A climatological spectral study of the 500mb geopotential height of the Northern Hemisphere. J. Atmos. Sci., 33, 1607-1623.
2. Booth J. F., Y.-K. Kwon, S. Ko, J. Small, R. Madsek, 2017: Spatial Patterns and Intensity of the Surface Storm Tracks in CMIP5 Models. Journal of Climate, 30, 4965–4981.
3. Chang, E., S. Lee and K. Swanson, 2002: Storm track dynamics. J. Climate, 15, 2163-2183.
4. Wallace, J.M., G-H Lim, M. L Blackmon, 1988: Relationship between cyclone tracks, anticyclone tacks and baroclinic waveguides. J. Atmos. Sci., 45, 439-462.

3.8 Multi-Case Example Diagnostic Documentation

Last update: Apr 2024

This POD illustrates how multiple cases (experiments) can be analyzed together. The muliple cases are specified to the MDTF Framework where they are initialized and preprocessed independently.

Note: This POD assumes familiarity with the single-case example diagnostic

3.8.1 Version & Contact info

- Version/revision information: version 1.1 (Oct 2022)
- Model Development Task Force Framework Team

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt). Unless you've distributed your script elsewhere, you don't need to change this.

3.8.2 Functionality

The framework generates yaml file called *case_info.yml* with environment variables for the file paths and variable information for each case. The **example_multicase** POD reads the yaml file information into a dictionary, and loops through the dictionary to read near-surface air temperature (TAS) data for each case. The POD time averages the data and calculates the anomaly relative to the global mean. The anomalies are zonally-averaged and the results from all cases are shown on a single plot.

3.8.3 Required programming language and libraries

- Python >= 3.11
- xarray
- numpy
- matplotlib
- yaml
- sys

3.8.4 Required model output variables

- tas - Surface (2-m) air temperature (CF: air_temperature)

3.8.5 References

1. E. D. Maloney et al. (2019): Process-Oriented Evaluation of Climate and Weather Forecasting Models. *BAMS*, **100** (9), 1665–1686, doi:10.1175/BAMS-D-18-0042.1[172].

3.9 Forcing Feedback Diagnostic Package

Last update: 12/21/2023

The Forcing Feedback Diagnostic package evaluates a model's radiative forcing and radiative feedbacks. This is a commong framework for understanding the radiative constraints on a model's climate sensitivity and is outlined in detail by *Sherwood et al. (2015)* (page 119), among many others. To compute radiative feedbacks, anomalies of temperature, specific humidity and surface albedo are translated into radiative anomalies by multiplying them by radiative kernels developed from the CloudSat/CALIPSO Fluxes and Heating Rates product (*Kramer et al. 2019* (page 119)).These radiative anomalies are regressed against the model's global-mean surface temperature anomalies to estimate the radiative feedbacks. Cloud radiative feedbacks are computed as the change in cloud radiative effects from the model's TOA radiative flux variables, corrected for cloud masking using the kernel-derived non-cloud radiative feedbacks (*Soden et al. 2008* (page 119)). The Instantaneous Radiative Forcing is computed first under clear-sky conditions by subtracting kernel-derivred clear-sky radiative feedbacks from the clear-sky TOA radiative imbalance diagnosed from the model's radiative flux variables. The all-sky Instantaneous Radiative Forcing is estimated by dividing the clear-sky Instantaneous Radiative Forcing by a cloud masking constant (*Kramer et al. 2021* (page 119)). All radiative quantities in this package are defined at the TOA and positive represents an increase in net downwelling or a radiative heating of the planet.

3.9.1 Contact info

- PI of the project: Brian Soden, University of Miami (bsoden@rsmas.miami.edu);
- Current developer: Ryan Kramer (ryan.kramer@noaa.gov)

Open source copyright agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.9.2 Functionality

The currently package consists of: - a Python script (forcing_feedback.py), which sets up the directories and calls... - ... an Python script (forcing_feedback_kernelcalcs.py) which reads the data, performs the calculations and saves the results to temporary netcdfs./././. - ... Finally, a Python script (forcing_feedback_plots.py) reads in the temporary results, observational radiative forcing and feedbacks and creates plots. Throughout the package, the scripts use Python functions defined in a third Python script (forcing_feedback_util.py)

[172] https://doi.org/10.1175/BAMS-D-18-0042.1

As a module of the MDTF code package, all scripts of this package can be found under `mdtf/MDTF_$ver/var_code/forcing_feedback` and pre-digested observational data and radiative kernels (in netcdf format) under `mdtf/inputdata/obs_data/forcing_feedback` Place your input data at: `mdtf/inputdata/model/$model_name/mon/`

3.9.3 Required programming language and libraries

Python is required to run the diagnostic.

The part of the package written in Python requires packages os, sys, numpy, xarray, scipy, matplotlib, cartopy and dask. These Python packages are already included in the standard Anaconda installation

3.9.4 Required model output variables

The following three 3-D (lat-lon-time), monthly model fields are required: - surface skin temperature ("ts" in CMIP conventions) - TOA incident shortwave radiation ("rsdt") - TOA outgoing all-sky shortwave radiation ("rsut") - TOA outgoing clear-sky shortwave radiation ("rsutcs") - TOA outgoing all-sky longwave radiation ("rlut") - TOA outgoing clear-sky longwave radiation ("rlutcs") - Surface downwelling all-sky shortwave radiation ("rsds") - Surface downwelling clear-sky shortwave radiation ("rsdscs") - Surface upwelling all-sky shortwave radiation ("rsus") - Surface upwelling clear-sky shortwave radiation ("rsuscs")

The following 4-D (lat-lon-level-time), monthly model fields are requied: - Air temperature ("ta" in CMIP conventions) - Specific humidity ("hus")

The observational estimates (see below) are for 2003-2014. The start date is based on data availability while the end date was selected to match the end date of relevant CMIP6 simulations. For an ideal comparison, the model data used in this POD should cover the same period and have realistic, historical forcing boundary conditions. However, this package will still have value as a "gut check" for any simulation, especially with respect to radiative feedbacks, which often exhibit similar characteristics regardless of the forcing scenario.

3.9.5 More about the diagnostic

a) Choice of reference dataset

While total TOA radiative changes are directly observed, the radiative feedback and radiative forcing components are not. Therefore, in this package the observational estimates of radiative feedbacks and radiative forcing are derived by multiplying data from ERA5 Reanalysis by the CloudSat/CALIPSO radiative kernels mentioned above. Global-mean surface temperature anomalies from ERA5 are used to compute the radiative feedbacks from the kernel-derived radiative anomalies as described above. To diagnose the instantaneous radiative forcing, the kernel-derived, clear-sky estimates of radiative feedbacks are subtracted by a measure of the total radiative anomalies at the TOA. For the observational dataset used here, that total radiative anomaly estimates is from CERES. The methods for diagnosing these radiative changes in observations are outlined by *Kramer et al. 2021* (page 119) and *He et al. 2021* (page 119)

3.9.6 References

1. Sherwood, S. C., Bony, S., Boucher, O., Bretherton, C., Forster, P. M., Gregory, J. M., & Stevens, B. (2015). Adjustments in the Forcing-Feedback Framework for Understanding Climate Change. *Bulletin of the American Meteorological Society*, **96** (2), 217–228. https://doi.org/10.1175/BAMS-D-13-00167.1

2. Kramer, R. J., Matus, A. V., Soden, B. J., & L'Ecuyer, T. S. (2019). Observation-Based Radiative Kernels From CloudSat/CALIPSO. *Journal of Geophysical Research: Atmospheres*, 2018JD029021. https://doi.org/10.1029/2018JD029021

3. Soden, B. J., Held, I. M., Colman, R., Shell, K. M., Kiehl, J. T., & Shields, C. A. (2008). Quantifying Climate Feedbacks Using Radiative Kernels. *Journal of Climate*, **21** (14), 3504–3520. https://doi.org/10.1175/2007JCLI2110.1

4. Kramer, R.J, He, H., Soden, B.J., Oreopoulos, R.J., Myhre, G., Forster, P.F., & Smith, C.J. (2021) Observational Evidence of Increasing Global Radiative Forcing. *Geophys. Res. Lett.*, **48** (7), e2020GL091585. https://doi.org/10.1029/2020GL091585

3.10 Mixed layer depth

Last update: 1/27/2021

This POD calculates maps of the mean mixed layer depth (MLD) for the period 2000-2014 for one model ensemble member.

Some CMIP6 models provide the variable `mlotst`, which is the mixed layer depth calculated instantaneously on the model timestep. To compare model output with monthly reanalysis products, we instead compute mixed layer depth from the monthly output salinity and temperature fields (`so` and `thetao`). We assume the salinity field `so` represents practical salinity.

We define the MLD as the depth where the density difference from the 10m depth value exceeds 0.03 kg m^-3 (de Boyer Montégut et al, 2004). Density is calculated from temperature and salinity using the Thermodynamic Equation Of Seawater - 2010 (TEOS-10) equation of state via the gsw package. The archived `mlotst` uses 0.125 kg m^-3 criterion according to the CMIP6 protocol for the instantaneous model fields, but we choose a higher criterion as the monthly fields are smoother than instantaneous fields. If this criteria is not exceeded in a given grid cell, we set the mixed layer depth to the ocean depth.

We compare model results to the EN4 reanalysis (Good et al. 2013). This product provides practical salinity and potential temperature, and the mixed layer depth is computed using the same approach as the models.

3.10.1 Version & Contact info

- Version 1 (1/26/2021)
- PI (Cecilia Bitz, University of Washington, bitz@uw.edu)
- Developer/point of contact (Lettie Roach, University of Washington, lroach@uw.edu)
- Other contributors: Cecilia Bitz

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

3.10.2 Functionality

The current package consists of one file:

1. mixed_layer_depth.py which loads data, calculates MLD, regrids to grid of observations and plots monthly means.

3.10.3 Required programming language and libraries

Python version 3, numpy, pandas, scipy, netCDF4, cftime, xarray, dask, esmpy, xesmf, matplotlib, cartopy, gsw

3.10.4 Required model output variables

Monthly mean salinity and temperature on vertical levels on the original model grid for 1979-2014 from a historical simulation. The vertical coordinate must be in units of meters.

3.10.5 References

1. Roach, L., C.M. Bitz, et al. In preparation.
2. de Boyer Montégut, C., G. Madec, A. S. Fisher, A. Lazar, and D. Ludicone (2004), Mixed layer depth over the global ocean: An examination of profile data and a profile-based climatology, J. Geophys. Res., 109, 20, doi: 10.1029/2004JC002378.
3. Good, S. A., M. J. Martin and N. A. Rayner, 2013. EN4: quality controlled ocean temperature and salinity profiles and monthly objective analyses with uncertainty estimates, Journal of Geophysical Research: Oceans, 118, 6704-6716, doi:10.1002/2013JC009067

3.11 MJO Propagation and Amplitude Diagnostic Package

Last update: 02/27/2019

This MJO propagation and amplitude diagnostic metrics is mainly motivated by recent multi-model studies that model skill in representing eastward propagation of the MJO is closely related to model winter mean low-level moisture pattern over the Indo-Pacific region, and the model MJO amplitude tends to be tightly associated with the moisture convective adjustment time scale. This package is designed to provide further independent verification of these above processes based on new GCM simulations.

3.11.1 Version & Contact info

- Version 2.0.
- PI and POC: Xianan Jiang (UCLA; xianan@ucla.edu)
- Developers: Alex Gonzalez (agon@iastate.edu), Xianan Jiang (xianan@ucla.edu)
- Contributors: E. Maloney (CSU), D. Waliser (JPL), Ming Zhao (GFDL)

Open source copyright agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.11.2 Functionality

The currently package (mjo_diag.py) consists of following functionalities:

1. Model skill scores of MJO eastward propagation versus winter mean low-level moisture pattern over Indo-Pacific;
2. Model amplitude of MJO over the Indian Ocean versus moisture convective adjustment time-scale.

As a module of the MDTF code package, all scripts of this package can be found under `mdtf/MDTF_$ver/var_code/mjo_diag`", and pre-digested observational data under `mdtf/inputdata/obs_data/mjo_diag`.

3.11.3 Required programming language and libraries

This package is mainly written in NCAR Command Language (NCL) which is driven by Python 2. A newer version of the NCL above 6.4 is recommended for implementation of this diagnostic package.

3.11.4 Required model output variables

The following 3D (lat-lon-time) or 4D (lat-lon-pressure-time) model fields are required:

1. 3D precipitation rate (units: mm d^{-1}; at least at daily interval or higher-frequency)
2. 4D specific humidity from 1000hPa to 100hPa (g g^{-1}; at least at daily interval or higher-frequency);

Daily 3D column water vapor (CWV, or precipitable water vapor; units: mm = kg m^{-2}) will be calculated from (2) for calculation of moisture convective time-scale.

3.11.5 References

1. Jiang, X. (2017), Key processes for the eastward propagation of the Madden-Julian Oscillation based on multi-model simulations, *JGR-Atmos*, **122**, 755–770, https://doi.org/10.1002/2016JD025955.
2. Gonzalez, A. O., and X. Jiang (2017), Winter mean lower tropospheric moisture over the Maritime Continent as a climate model diagnostic metric for the propagation of the Madden-Julian oscillation, *Geophys. Res. Lett.*, **44**, 2588–2596, https://doi.org/10.1002/2016GL072430.
3. Jiang, X., M. Zhao, E. D. Maloney, and D. E. Waliser, 2016: Convective moisture adjustment time scale as a key factor in regulating model amplitude of the Madden-Julian Oscillation. *Geophys. Res. Lett.*, **43**, 10,412-410,419.

3.11.6 More details about this diagnostic

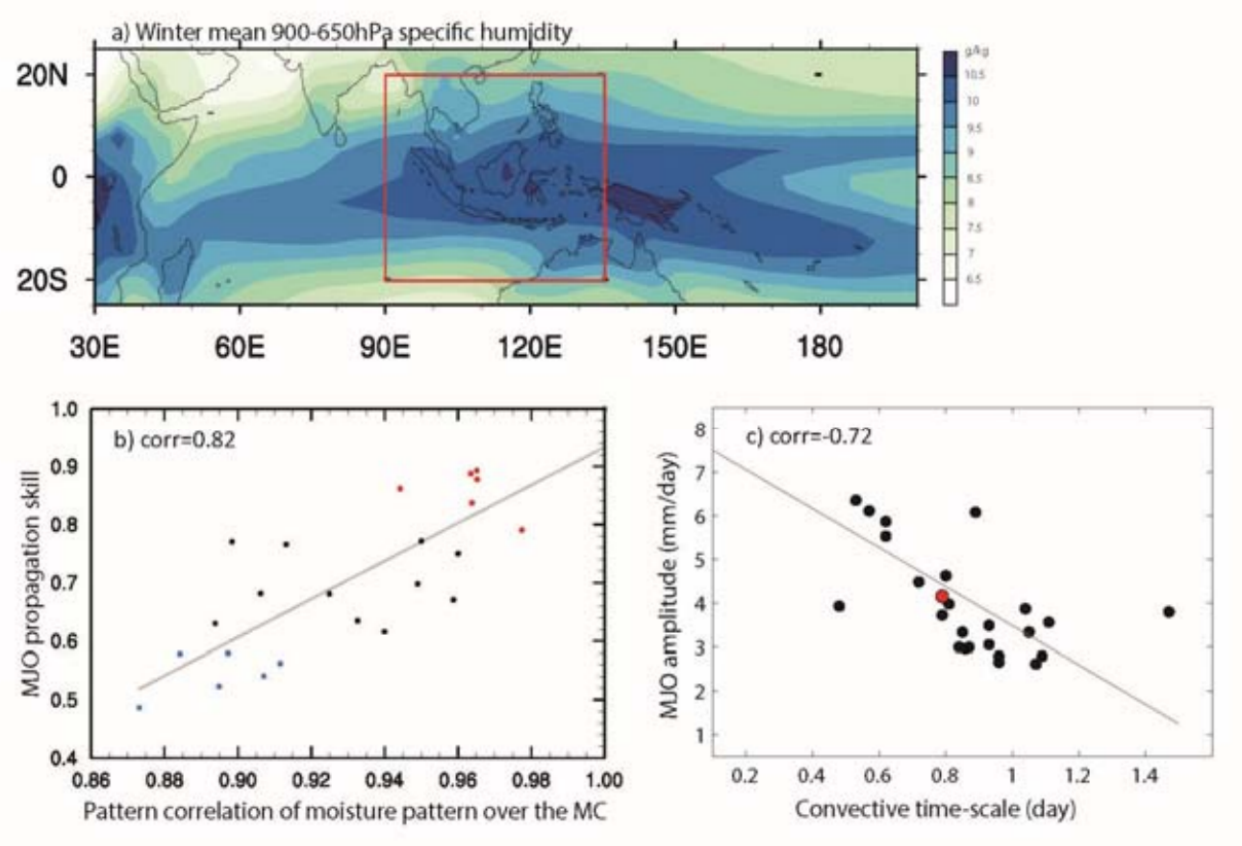


Fig. 5: a) Winter (Nov-Apr) mean 650-900hPa specific humidity based on ERA-Interim reanalysis; b) Scatter plot of model skill for eastward propagation of the MJO versus model skill of the mean 650-900hPa moisture pattern over the Maritime Continent (MC; red rectangle in a) based on multi-model simulations from the MJOTF/GASS project. Model MJO propagation skill is defined by the pattern correlation of Homvöller diagrams of model simulated rainfall anomalies associated with the MJO against its observed counterpart following *Jiang et al. (2015)* (page 119). Red (blue) dots denote good (poor) MJO models. c) Scatter plot of MJO amplitude and model convective moisture adjustment time scale in models (black dots) and observations (red dots). The MJO amplitude in each model is defined by the standard deviation of 20-100 day band-pass filtered rainfall over the Indian Ocean (75-85oE; 10oS-10oN) during winter. Convective time scale in a model is defined by the ratio of precipitable water (W) anomaly to precipitation (P) anomaly associated with the MJO and derived by a regression approach. Before conducting the regression, both W and P anomalies are subject to 20-100 day filtering and averaged over the Indian Ocean (75-85oE; 10oS-10oN) region. Adapted from *Jiang et al. (2016)* (page 119) and *Gonzalez and Jiang (2017)* (page 119).

This diagnostic metric for MJO propagation is motivated by findings that the horizontal advection of column moist static energy, or equivalently the lower-tropospheric moisture, plays a critical role in driving the eastward propagation of the winter MJO (e.g., *Maloney 2009* (page 75); *Maloney et al. 2010* (page 75); *Kiranmayi and Maloney 2011* (page 75); *Sobel et al. 2014* (page 76); *Chikira 2014* (page 56); *Kim et al. 2014* (page 75); *Adames and Wallace 2015* (page 119); *Jiang 2017* (page 75); *Kim et al. 2017* (page 75)). Under this process, the spatial distribution of the winter mean lower-tropospheric moisture distribution over the equatorial Indo-Pacific region (Fig. 1a) is crucial for moistening (drying) to the east (west) of the MJO convection through advection by MJO anomalous winds. The critical role of the mean lower-tropospheric moisture pattern for the MJO eastward propagation is confirmed by multi-model simulations from the MJO Task Force / GEWEX GASS MJO model comparison project (*Jiang 2017* (page 75); *Gonzalez and Jiang 2017* (page 119)). In particular, model skill in representing the 900-650hPa mean moisture pattern over the Maritime Continent region (red rectangle in Fig. 1a) exhibits a high correlation (about 0.8) with model MJO

eastward propagation skill in about 25 GCM simulations (Fig. 1b).

On the other hand, the convective moisture adjustment time scale () in a model, defined by the ratio of intraseasonal perturbations of precipitable water and surface precipitation (e.g., *Bretherton et al. 2004* (page 119); *Peters and Neelin 2006* (page 76); *Sobel and Maloney 2013* (page 76)), is selected as a diagnostic metric for model MJO amplitude, which is motivated by the high anti-correlation (-0.72) between and model MJO amplitude across multi-model simulations as indicated in Jiang et al. (2016, Fig. 1c). The convective moisture adjustment time scale depicts how rapidly precipitation must occur to remove excess column water vapor, or alternately the efficiency of surface precipitation generation per unit column water vapor anomaly, and is highly relevant to the convection onset diagnostics described above.

With this diagnostic packet, relationships between model skill in representing MJO eastward propagation and winter mean lower-tropospheric moisture as shown in Fig. 1b, and model MJO amplitude and moisture convective adjustment time scale as in Fig. 1c are examined based on specified model simulations.

3.11.7 References

4. Adames, Á. F. and J. M. Wallace, 2015: Three-Dimensional Structure and Evolution of the Moisture Field in the MJO. *J. Atmos. Sci.*, **72**, https://doi.org/10.1175/JAS-D-15-0003.1, 3733-3754.

5. Bretherton, C. S., M. E. Peters, and L. E. Back, 2004: Relationships between water vapor path and precipitation over the tropical oceans. *J. Clim.*, **17**, 1517-1528.

6. Chikira, M., 2014: Eastward-Propagating Intraseasonal Oscillation Represented by Chikira–Sugiyama Cumulus Parameterization. Part II: Understanding Moisture Variation under Weak Temperature Gradient Balance. *J. Atmos. Sci.*, **71**, https://doi.org/10.1175/JAS-D-13-038.1, 615-639.

7. Gonzalez, A. O. and X. Jiang, 2017: Winter Mean Lower-Tropospheric Moisture over the Maritime Continent as a Climate Model Diagnostic Metric for the Propagation of the Madden-Julian Oscillation. *Geophys. Res. Lett.*, https://doi.org/10.1002/2016GL072430.

8. Jiang, X., 2017: Key processes for the eastward propagation of the Madden-Julian Oscillation based on multi-model simulations. *Journal of Geophysical Research: Atmospheres*, https://doi.org/10.1002/2016JD025955.

9. Jiang, X., M. Zhao, E. D. Maloney, and D. E. Waliser, 2016: Convective moisture adjustment time scale as a key factor in regulating model amplitude of the Madden-Julian Oscillation. *Geophys. Res. Lett.*, **43**, https://doi.org/10.1002/2016GL070898, 10,412-10,419.

10. Kim, D., J.-S. Kug, and A. H. Sobel, 2014: Propagating versus Nonpropagating Madden–Julian Oscillation Events. *J. Clim.*, **27**, https://doi.org//10.1175/JCLI-D-13-00084.1, 111-125.

11. Kim, D., H. Kim, and M.-I. Lee, 2017: Why does the MJO detour the Maritime Continent during austral summer? *Geophys. Res. Lett.*, https://doi.org//10.1002/2017GL072643, n/a-n/a.

12. Kiranmayi, L. and E. D. Maloney, 2011: Intraseasonal moist static energy budget in reanalysis data. *Journal of Geophysical Research: Atmospheres*, **116**, https://doi.org/10.1029/2011JD016031, D21117.

13. Maloney, E. D., 2009: The Moist Static Energy Budget of a Composite Tropical Intraseasonal Oscillation in a Climate Model. *J. Clim.*, **22**, 711-729.

14. Maloney, E. D., A. H. Sobel, and W. M. Hannah, 2010: Intraseasonal variability in an aquaplanet general circulation model. *Journal of Advances in Modeling Earth Systems*, **2**, https://doi.org/10.3894/james.2010.2.5.

15. Peters, O. and J. D. Neelin, 2006: Critical phenomena in atmospheric precipitation. *Nat Phys*, **2**, 393-396.

16. Sobel, A. and E. Maloney, 2013: Moisture Modes and the Eastward Propagation of the MJO. *J. Atmos. Sci.*, **70**, https://doi.org/10.1175/Jas-D-12-0189.1, 187-192.

17. Sobel, A., S. Wang, and D. Kim, 2014: Moist Static Energy Budget of the MJO during DYNAMO. *J. Atmos. Sci.*, **71**, https://doi.org/10.1175/JAS-D-14-0052.1, 4276-4291.

3.12 MJO Suite Diagnostic Module From NCAR

Last update: 03/11/2019

This module computes many of the diagnostics described by the the US-CLIVAR Madden-Julian Oscillation (MJO) working group and developed by NCAR's Dennis Shea for observational data. Using daily precipitation, outgoing longwave radiation, zonal wind at 850 and 200 hPa and meridional wind at 200hPa, the module computes anomalies, bandpass-filters for the 20-100 day period, calculates the MJO Index as defined as the running variance over the band-pass filtered data, performs an EOF analysis, and calculates lag cross-correlations, wave-number frequency spectra and composite life cycles of MJO events.

3.12.1 Contact info

- PI: Rich Neale, NCAR
- Current Developer: Dani Coleman (bundy@ucar.edu), NCAR
- Contributors: Dennis Shea, Andrew Gettleman, Jack Chen (NCAR)

Open source copyright agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.12.2 Functionality

All scripts can be found at: `mdtf/MDTF_$ver/var_code/MJO_suite`

1. Read in daily model output: daily_netcdf.ncl
2. Computes daily anomalies: daily_anom.ncl
3. Calculates EOFs: mjo_EOF.ncl
4. Creates MJO lag plots: mjo_lag_lat_lon.ncl
5. Calculates MJO spectra: mjo_spectra.ncl
6. Calculates principle component from EOF analysis: mjo_EOF_cal.ncl
7. MJO life cycle composite: mjo_life_cycle_v2.ncl

Preprocessed observational data from NCEP, ERA and TRMM are located in
`mdtf/inputdata/obs_data/MJO_suite`.
Place your input data at: `mdtf/inputdata/model/$model_name/day`
index.html can be found at: `mdtf/MDTF_$ver/wkdir/MDTF_$model_name`

3.12.3 Required Programing Language and libraries

All these scripts required NCAR Command Language Version 6.3.0 or higher
The following Python packages are required: os, glob

3.12.4 Required input data to the module

The following five 3-D (lat-lon-time) model fields are required with a daily time output

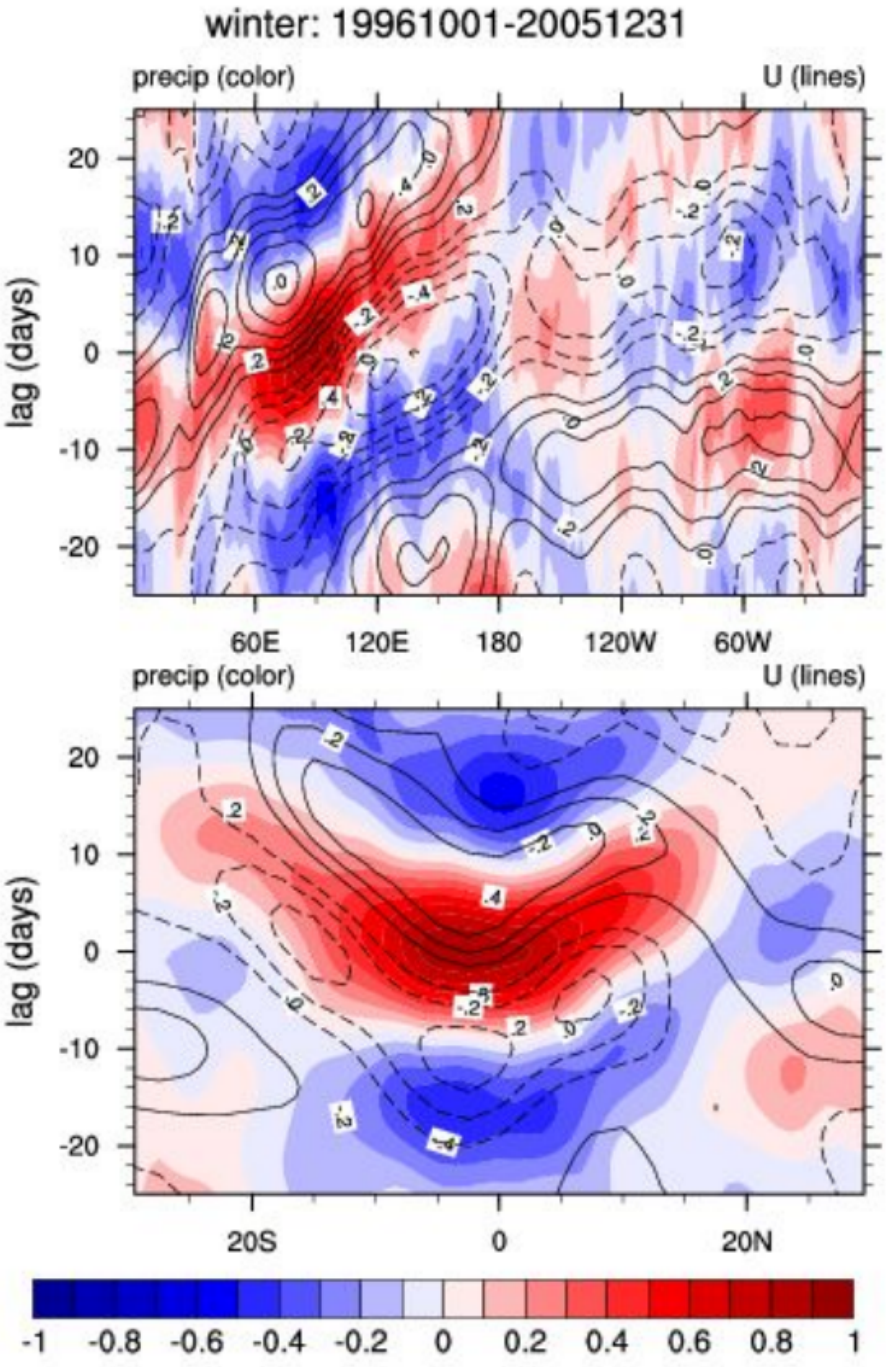
1. Precipitation rate (units: mm/s = kg/m^2/s) or mm/day with appropriate conversion
2. Outgoing Longwave radiation (units: W/m^2)
3. Zonal wind at 850hPa (units: m/s)
4. Zonal wind at 200hPa (units: m/s)
5. Meridional wind at 200hPa (units:m/s)

3.12.5 References

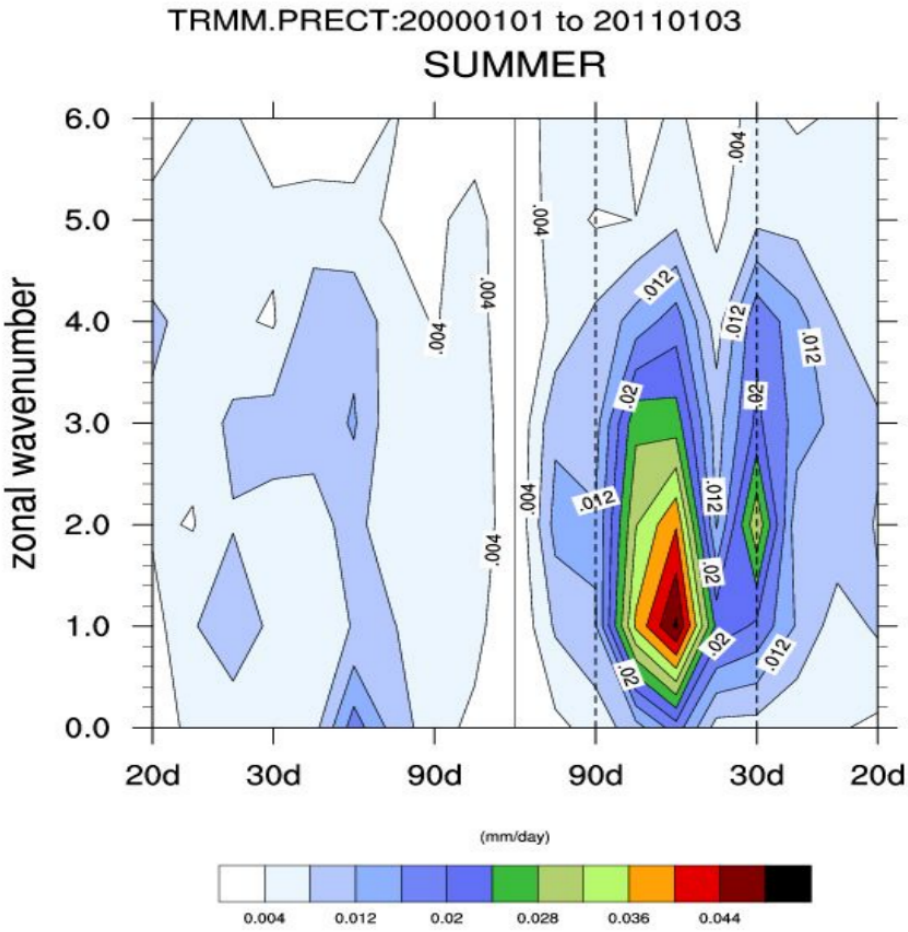
1. MJO Simulation Diagnostics, Waliser et al, 2009, *J.Clim.*, **22**: 3006-3030, https://doi.org/10.1175/2008JCLI2731.1

3.12.6 More About the Diagnostic

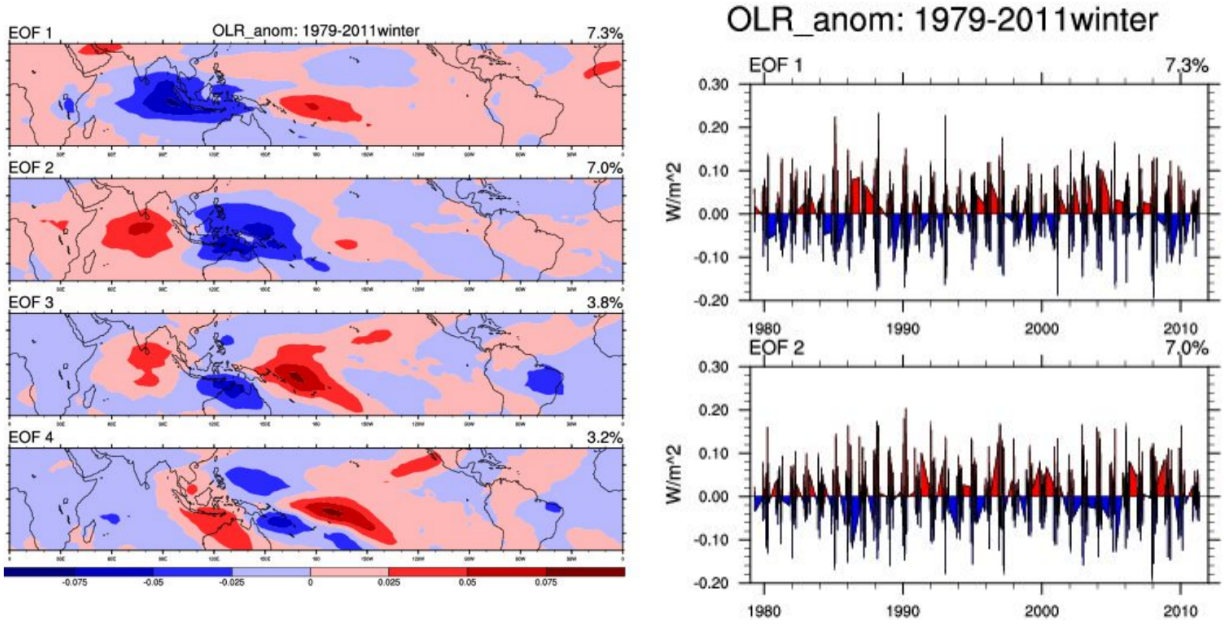
An extensive explanation of the figures and techniques used to achieve them can be found on Dennis Shea's NCL MJO CLIVAR page. Some examples are provided below:



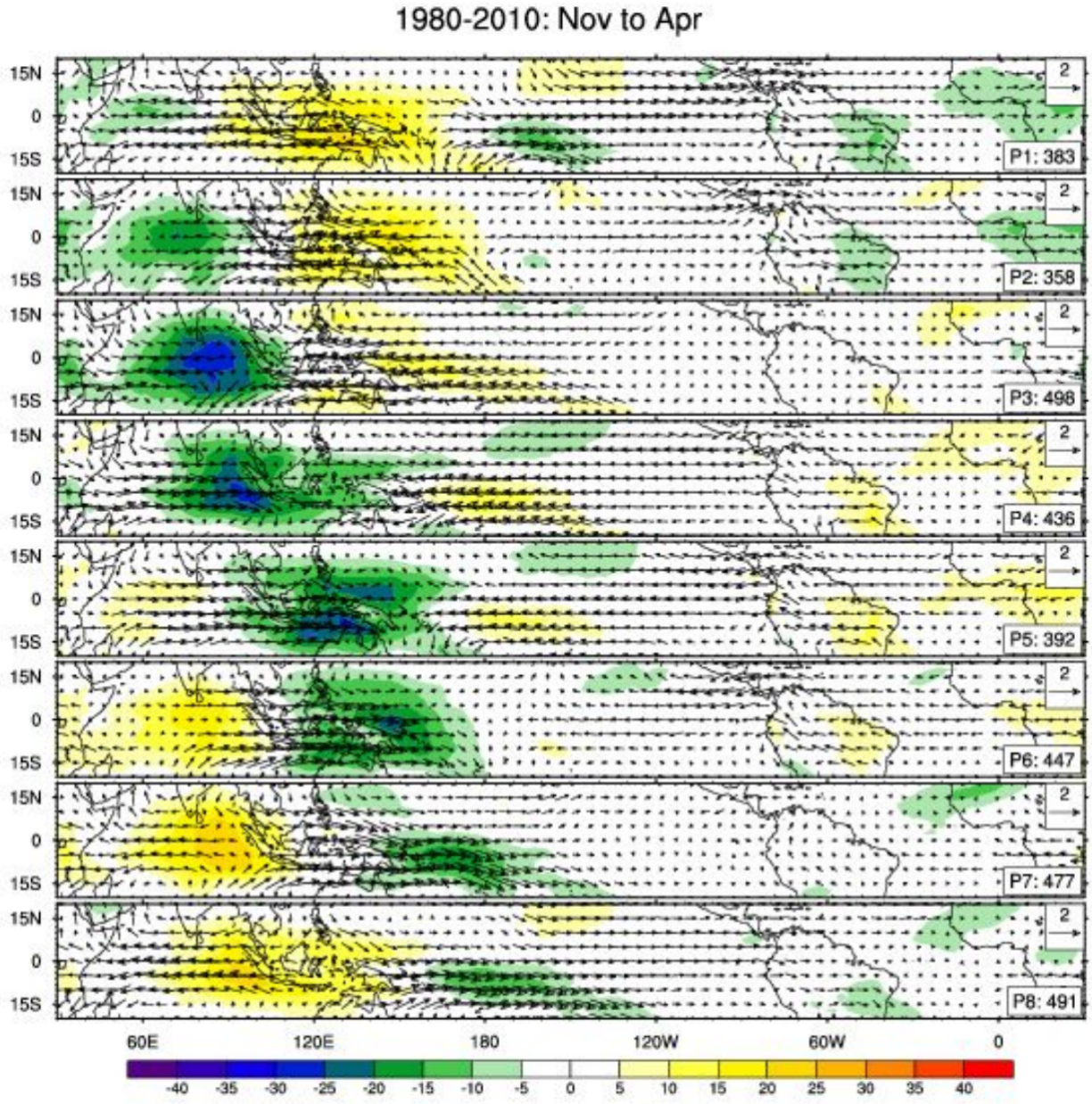
Lag Correlation: Lag correlation diagram using on 20-100 day band-pass filtered daily data. The reference time series is the central Indian Ocean regional precipitation time series, which is correlated with precipitation and zonal wind anomalies in specified regions at different lags. Lag-longitude and lag-latitude plots of correlation values for different regions are shown. Color is for precipitation correlations while the lagged correlations for the zonal winds are the contours. These are analogous the Figures 5 and 6 in the reference article except they are for one season.



Wavenumber-Frequency Spectra: The wavenumber - frequency spectra for each season, with vertical reference lines indicating at for 30 and 80 days.



EOF analysis (univariate): Conventional (covariance) univariate EOF analysis for 20-100 day band-pass filtered data of outgoing longwave radiation over 30S to 30N.



Composite Life-cycles: The first and second principle components of the EOF analysis are used to derive the appropriate MJO phase category. The size of the reference anomaly wind vector is in the upper right. The phase (eg P3, means "Phase 3") and the number of days used to create the composite are at the lower right.

3.13 MJO Teleconnection Diagnostic Package

Last Update: 2/1/2019

The teleconnection diagnostics first generate maps of MJO phase composites of 250 hPa geopotential height and precipitation for observations and several CMIP5 models, putting behavior of the candidate model within this cloud of models and observations. Then, average teleconnection performance across all MJO phases defined using a pattern correlation of geopotential height anomalies is assessed relative to 1) MJO simulation skill and 2) biases in the North Pacific jet zonal winds to determine reasons for possible poor teleconnections. Performance of the candidate model is assessed relative to a cloud of observations and CMIP5 simulations.

3.13.1 Contact info

- PI: Eric D. Maloney (eric.maloney@colostate.edu), Colorado State University
- Current Developer: Bohar Singh (bohar.singh@colostate.edu), Colorado State University
- Contributors: Stephanie Henderson (University of Wisconsin–Madison), Bohar Singh (CSU)

Open source copyright agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.13.2 Functionality

1. Calculation of RMM indices for a new model will be saved in `wkdir/casename/MJO_teleconnection/model/netCDF` in txt format (mjo_diag_RMM_MDTF.ncl)
2. Z250 phase composite for all MJO phases (mjo_diag_geop_hgt_comp_MDTF.ncl)
3. Pattern correlation with observation (ERA-I Z250) (mjo_diag_Corr_MDTF.ncl)
4. Precipitation (30S-30N) phase composite for all MJO phases (mjo_diag_prec_comp_MDTF.ncl)
5. Extended winter wave number-frequency power spectrum of precipitation to get the ratio of eastward and westward propagation power (mjo_diag_EWR_MDTF.ncl)
6. Area averaged DJF mean U250 error (model-observation) over Pacific Ocean (15N80N,120E-120W) (mjo_diag_U250_MDTF.ncl)
7. ncl script to plot teleconnection skill v/s MJO skill (mjo_diag_fig1_MDTF.ncl)
8. ncl script to plot teleconnection skill v/s basic state skill (mjo_diag_fig1_MDTF.ncl)

All scripts can be found at: `mdtf/MDTF_$ver/var_code/MJO_teleconnection`

Preprocessed data for 10 CMIP5 models and observations data is located at``mdtf/MDTF_$ver/obs_data/MJO_teleconnection``
Keep your input data at: `mdtf/MDTF__$ver/$model_name/day`
Index.html can be found at: `mdtf/MDTF_$ver/ wkdir/MDTF_$model_name`

3.13.3 Required Programing Language and libraries

All these scripts required NCAR Command Language Version 6.3.0 or higher in addition to ncl
The following Python packages are required: os, glob, json, Dataset, numpy, scipy, matplotlib & networkx, warnings, numba, netcdf4
Please use Anaconda: These Python packages are already included in the standard installation

3.13.4 Required input data to the module

The following five 3-D (lat-lon-time) model fields are required:

1. precipitation rate (units: mm/s = kg/m^2/s) or mm/day with appropriate conversion, daily avg
2. Outgoing Longwave radiation (units: W/m^2)
3. U850 wind (units: m/s)
4. U250 wind (units: m/s) (Note: U250 wind is used instead of u200 for RMM index calculation)
5. Z250 (units:m)

Please change the variable names and conversion factor corresponding to your data before running the MJO teleconnection diagnostics at: `var_code/util/set_variables_CESM.py`
Please condense each input variable into a single file

3.13.5 References

1. Henderson, S. A., Maloney, E. D., & Son, S. W. (2017). Madden–Julian oscillation Pacific teleconnections: The impact of the basic state and MJO representation in general circulation models. *Journal of Climate*, **30** (12), 4567-4587.

3.13.6 More About the Diagnostic

Henderson et al (2017) (page 119) documented reasons for MJO midlatitude teleconnection errors in CMIP5 models. Since MJO teleconnections have significant impacts on atmospheric rivers, blocking, and other extreme events in the midlatitudes, teleconnection errors in models have important implications for the subseasonal prediction of midlatitude weather extremes (e.g. Henderson et al. 2016; Mundhenk et al. 2018; Baggett et al. 2017). In addition to extended analyses of stationary wavenumber biases and use of a linear baroclinic model to diagnose CMIP model biases, *Henderson et al (2017)* (page 119) developed diagnostics linking teleconnection biases to biases in the position and extent of the North Pacific jet.

The first diagnostic in this POD presents MJO composite 250 hPa geopotential height anomalies for ERA-I, the candidate model (upper right), and six other CMIP5 models assessed to have good MJO performance. All composites are generated as a function of MJO phase as defined according to Wheeler and Hendon (2004). An example of this diagnostic is presented in Figure 1 for phase 1 of the MJO.

The diagnostic next assesses teleconnection performance versus measures of basic state fidelity and MJO skill. Figure 2 from *Henderson et al (2017)* (page 119) contains two panels, each having MJO teleconnection performance during December-February on the y-axis. In Figure 2a, the x axis represents an MJO skill metric. While Figure 2a shows a relationship between MJO skill and teleconnection performance, even models with a good MJO can have poor teleconnection performance. For only the models assessed to have a sufficiently good MJO, Figure 2b assesses the relationship

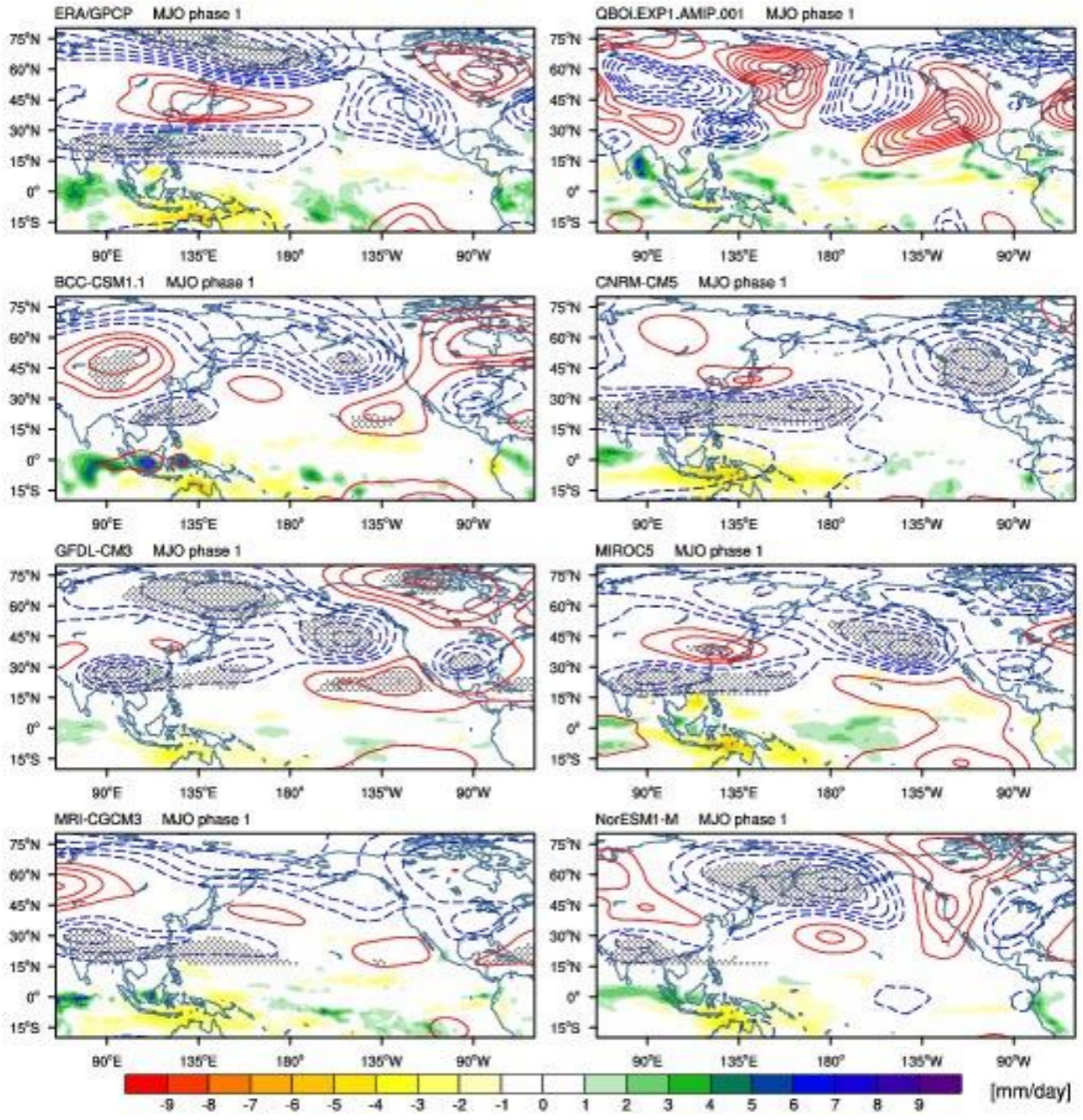


Fig. 6: Figure 1. MJO phase 1 pentad composites of anomalous 250-hPa geopotential height, where a pentad denotes a 5-day mean, in this case the average of lag days 0 - 4 following an MJO phase. ERA-I is shown in the upper left, and the candidate model in the upper right. Positive geopotential height anomalies are in red solid contours, and negative anomalies are in blue dashed contours. Contours are plotted every 10 m, and the zero contour is omitted. Anomalies found to be 95% significantly different from zero are dotted. The color shading shows the anomalous tropical precipitation composite during MJO phase 1.

between teleconnection performance and biases in the North Pacific zonal flow. Plus signs are a measure of the total root mean squared (RMS) error of the 250 hPa zonal flow over the region 15°N – 60°N, 110°E – 120°W, and the filled circle provides a measure of the RMS error in the length of the North Pacific subtropical jet. Both measures are correlated with MJO teleconnection performance, although biases in the jet provides a somewhat better metric (r =-0.7 versus -0.6 for the total RMS). Subsequent analysis showed that models with a jet that extends too far east tend to have degraded teleconnection performance. Model physics appears to play a key role in the extent of the Pacific jet, as was demonstrated by Neelin et al. (2016) in diagnosing projected California precipitation changes between CMIP3 and CMIP5 models into the late 21st Century. The Pod developed here places the candidate model in question into the cloud of other models on Figure 2, with separate links on the POD site for left and right panels of Figure 2.

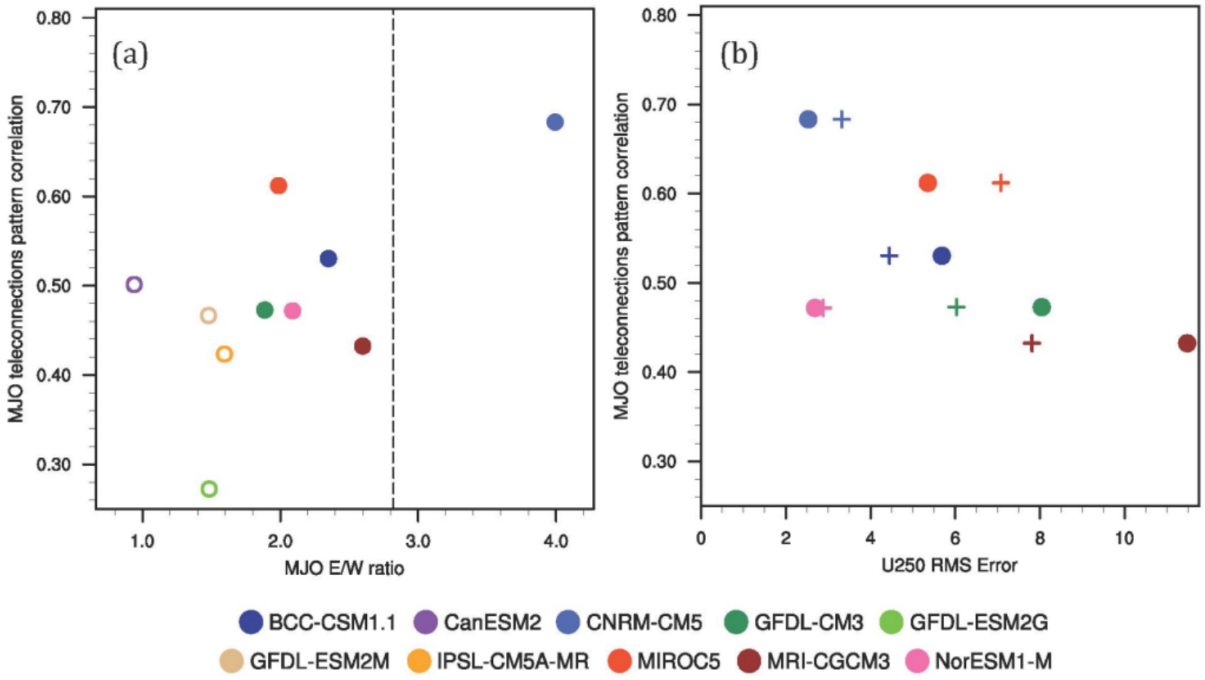


Fig. 7: Figure 2. From *Henderson et al (2017)* (page 119). December-February teleconnection performance averaged across all MJO phases (y-axis) versus (a) MJO skill (MJO E/W ratio) and (b) the RMS error of the 250-hPa December-February zonal wind. The observed E/W precipitation (GPCP) ratio is provided (dashed line, left panel). The MJO skill is derived as the ratio of eastward to westward power of equatorial precipitation in the 30-60 day, zonal wavenumbers 1-3 band (e.g. Ahn et al. 2017). Teleconnection performance was determined as pattern correlation of North Pacific and North America (15°S - 80°N, 130°E - 60°W) MJO composite 250 hPa geopotential height anomalies between CMIP models and ERA-i reanalysis averaged over all MJO phases. In panel (b), the crosses show the model 250hPa zonal wind RMS error over the full Pacific basin, while the closed circles indicate the longitudinal RMS error of the subtropical jet.

3.14 Ocean Surface Latent Heat Flux Diagnostic Documentation

Last update: 12/10/2021

Tropical intra-seasonal (20-100 day) convection regulates weather patterns globally through extratropical teleconnections. Surface latent heat fluxes help maintain tropical intra-seasonal convection and the Madden-Julian oscillation by replenishing column water vapor lost to precipitation. Latent heat fluxes estimated using surface meteorology from moorings or satellites and the COARE3.0 bulk flux algorithm suggest that latent heat fluxes contribute about 8% of the intra-seasonal precipitation anomaly over the Indian and western tropical Pacific Oceans [Dellaripa and Maloney, 2015, Bui et al., 2020].

For this diagnostic, we use in-situ data from TAO/TRITON/RAMA to create a location-based latent heat flux matrix determined by specific humidity deficiency at the surface layer (dq) and surface wind speed (sfcWind). By comparing the matrix between observation and models/reanalysis, the diagnostic can help revealing where model/reanalysis latent heat flux biases are in the dq-sfcWind space. The latent heat flux biases shown in the diagnostic demonstrate dependence on both sfcWind and dq. An offline latent heat bias correction can be performed on simulations based on the bias latent heat fluxes matrix as a function of dq and sfcWind.

3.14.1 Version & Contact info

- Version/revision information: version 2 (12/10/2021)
- PI (Charlotte A. DeMott, Colorado State University, charlotte.demott@colostate.edu)
- Developer/point of contact (Chia-Wei Hsu, Colorado State University, Chia-Wei.Hsu@colostate.edu)

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

3.14.2 Functionality

The main script generates the Latent heat flux matrix and bias matrix.

Python function used

- **groupby_variables.bin_2d**
  [The function is written to bin the variable (target_var) in a xr.Dataset] based on two other variables (bin1_var, bin2_var) in the same xr.Dataset. The function calculate the mean, std, and count values of the target_var after binning.
- **model_read.regional_var**
  [The function is written to read the model output and required varaibles.] The function also crop the data based on the user set time period and region. Two varibales is calculated in this function 1) saturation specific humidity near surface (determined by surface temperature and surface pressure) and 2) dq which represent the vertical difference of specific humidity near surface.
- **obs_data_read.tao_triton**
  [The function is written to read the observational data and required varaibles] from the TAO/TRITON array. The function also crop the data based on the user set time period and region. Two varibales is calculated in this function 1) saturation specific humidity near surface (determined by surface temperature and surface pressure) and 2) dq which represent the vertical difference of specific humidity near surface.
- **obs_data_read.rama**
  [The function is written to read the observational data and required varaibles] from the RAMA array. The function also crop the data based on the user set time period and region. Two varibales is calculated in this function 1) saturation specific humidity near surface (determined by surface temperature and surface pressure) and 2) dq which represent the vertical difference of specific humidity near surface.

3.14.3 Required programming language and libraries

The programming language is python version 3 or up. The third-party libraries include “matplotlib”, “xarray”, “metpy”,”numpy”,”scipy”.

3.14.4 Required model output variables

With daily frequency from the model output. This diagnostic needs

input atmosphere model variables

1. ‘huss’ : Surface 2m Humidity (kg kg-1)
2. ‘ts’ : Skin Temperature (SST for open ocean; K)
3. ‘sfcWind’ : Near-Surface Wind Speed (10 meter; m s-1)
4. ‘psl’ : Sea Level Pressure (Pa)
5. ‘hfls’ : Surface Upward Latent Heat Flux (W m-2 and positive upward)
6. ‘pr’ : Precipitation (kg m-2 s-1)

The script is written based on the CESM2-CMIP6 daily data download hosted by WCRP.

The dimension of all variable is 3-D with (time,lat,lon) in dimension and 2-D array for lat and lon as coordinate.

3.14.5 Required observational data

With daily frequency from the observational data. This diagnostic needs

input observational variables

1. ‘RH’ : Relative Humidity (%)
2. ‘SST’ : Sea Surface Temperature (for open ocean; K)
3. ‘WindSpeed10m’ : Near-Surface Wind Speed (10 meter; m s-1)
4. ‘SLP’ : Sea Level Pressure (Pa)
5. ‘Latent’ : Surface Upward Latent Heat Flux (W m-2 and positive upward)
6. ‘airT’ : Near Surface Temperature (K)

data access :

All variables can be downloaded from PMEL NOAA hosted website https://www.pmel.noaa.gov/tao/drupal/flux/index.html

3.14.6 References

1. C.-W. Hsu et al. (2021): Ocean Surface Flux Algorithm Effects on Tropical Indo-Pacific Intraseasonal Precipitation. *GRL*, under review.

3.14.7 More about this diagnostic

Surface latent heat flux from ocean to the atmosphere is one of the important processes that provides water vapor and energy to the daily tropical rainfall. A visually intuitive latent heat flux diagnostic is proposed to better understand the model shortfall on its latent heat flux representation. This diagnostic allows a simple assessment of model latent heat flux biases arising either from biases in water vapor or surface wind speed as well as other empirical coefficients in the model. Sample POD result shows that, compared to ''observed" fluxes also estimated from water vapor and surface wind speed measured at tropical moorings, tropical latent heat fluxes in the NCAR CEMS2 models are significantly overestimated when extreme water vapor or surface wind speed happens.

3.15 Precipitation Buoyancy Diagnostic Package

The precipitation-buoyancy diagnostics POD is used to assess the thermodynamic sensitivity of model precipitation fields.

3.15.1 Scientific basis

Observations show that over tropical oceans, a lower tropospheric buoyancy metric B_L has a strong relationship to precipitation (*Ahmed and Neelin 2018* (page 88), *Ahmed et al. 2020* (page 88)). This buoyancy metric can further be decomposed into two components:

1. A measure of undilute buoyancy termed CAPE $_{\mathrm{L}}$, which measures the difference between boundary layer moist enthalpy and the free-tropospheric temperature. If convection were non-entraining, this would be the dominant thermodynamic measure affecting precipitation.
2. A measure of lower-free tropospheric sub-saturation SUBSAT $_{\mathrm{L}}$, which is computed as a departure from saturation in the lower free-troposphere. The influence of entrainment on convection is expressed through this measure.

In observations (ERA re-analysis and TRMM precipitation), precipitation appears to about equally sensitive to CAPE $_{\mathrm{L}}$ and SUBSAT $_{\mathrm{L}}$. However, climate models can show diverging behavior. To measure this relative sensitivity of precipitation to CAPE $_{\mathrm{L}}$ and SUBSAT $_{\mathrm{L}}$, a vector γ_{CS} is introduced. This has a direction that is expressed in degrees and takes values ranging from 0 to 90.

3.15.2 Version & Contact info

- Fiaz Ahmed (UCLA)
- PI: J. David Neelin (UCLA; neelin@atmos.ucla.edu)
- Current developer: Fiaz Ahmed

Open source copyright agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.15.3 Functionality

The currently package consists of following functionalities:

1. Precipitation Buoyancy curve and surface

As a module of the MDTF code package, all scripts of this package can be found under the *precipitaton_buoyancy_diag*

3.15.4 Required programming language and libraries

The is package is written in Python 3.7, and requires the following Python packages: numpy, scipy, matplotlib, cython, numba, & xarray. These Python packages are already included in the standard Anaconda installation.

3.15.5 Required model output variables

The following high-frequency model fields are required:

1. Precipitation rate
2. Vertical profile of temperature
3. Vertical profile of specific humidity
4. Surface pressure (optional)

3.15.6 References

1. Ahmed, F., & Neelin, J. D. (2018). Reverse engineering the tropical precipitation–buoyancy relationship. Journal of the Atmospheric Sciences, 75(5), 1587-1608.`__.
2. Ahmed, F., Adames, Á. F., & Neelin, J. D. (2020). Deep convective adjustment of temperature and moisture. Journal of the Atmospheric Sciences, 77(6), 2163-2186.`__.

3.15.7 More about this diagnostic

3.16 Phase and Amplitude of Precipitation Diurnal Cycle

Last update: 03/11/2019

The diurnal cycle package generates a simple representation of the phase (in local time) and amplitude (in mm/day) of total precipitation, comparing a lat-lon model output of total precipitation with observed precipitation derived from the Tropical Rainfall Measuring Mission (TRMM: https://pmm.nasa.gov/TRMM/mission-overview) satellite derived 3B42 product.

3.16.1 Version & Contact info

- Version 1:
- Current Developer: Rich Neale

Open source copyright agreement

This package is provided under the LGPLv3 license (see LICENSE.txt).

3.16.2 Functionality

a) Computation of the diurnal cycle (local time phase) and amplitude of the first diurnal harmonic of total precipitation from TRMM (a 3-hourly product) with 3-hourly mean or instantaneous output from a model b) Plotting of a lat lon field of phase (with a cyclic color label) and hue (saturation of phase color) for a number of pre-specified regions and seasons. c) Computation of the variance explained by the diurnal harmonic on a gridded lat lon d) Computation of the mean precipitation over the analysis period on a gridded lat lon

3.16.3 Required programming language and libraries

Requires NCAR Command Language (NCL) and associated numerical and graphical capabilities. License agreement https://www.ncl.ucar.edu/Download/NCL_binary_license.shtml and licenses therewithin.

3.16.4 Required model output variables

Code requires the input of total precipitation either as a single variable or one that can be derived from a convective and large-scale combination.

3.16.5 References

1. Gervais, M., J. R. Gyakum, E. Atallah, L. B. Tremblay, and R. B. Neale, 2014: How Well Are the Distribution and Extreme Values of Daily Precipitation over North America Represented in the Community Climate System Model? A Comparison to Reanalysis, Satellite, and Gridded Station Data. *Journal of Climate*, **27**, 5219–5239, https://doi.org/10.1175/jcli-d-13-00320.1.
2. Gettelman, A., P. Callaghan, V. E. Larson, C. M. Zarzycki, J. T. Bacmeister, P. H. Lauritzen, P. A. Bogenschutz, and R. B. Neale, 2018: Regional Climate Simulations With the Community Earth System Model. *Journal of Advances in Modeling Earth Systems*, **10**, 1245–1265, https://doi.org/10.1002/2017ms001227.

3.16.6 More about this diagnostic

This diagnostic set provides an analysis of the variation in diurnal peak of rainfall over both land and ocean. The land analysis in particular gives insight into the mechanisms of rainfall production during the day. A recognized bias of many climate models is that they produce a peak in rainfall that is too early in the day (~noon) when the peak should be later in the evening. This is frequently interpreted as being due to shortcomings in the representations of the atmospheric physics and land-atmosphere interactions. Additionally, there is more detail relating to the impact of orographic and coastal regions that can amplify, weaken and in a number of other ways, modify the precipitation diurnal cycle.

3.17 Sea Ice Suite

Last update: 1/27/2021

This POD calculates maps of the sea ice concentration (SIC) for the period 1979-2014 for one ensemble member from a model compared to observations for the following quantities:

- **mean**: the mean (obviously) by month $\%$.
- **trend**: the linear trend by month $\%s^{-1}$.
- **standard deviation**: the standard deviation by month $\%$.
- **standard deviation after detrending**: the standard deviation of detrended data by month $\%$.
- **one-lag correlation**: the correlation at a lag of one month and one year of detrended data by month . For a one-month lag, the map for January shows the correlation of January and February. The map for February shows the correlation of February and March. And so forth. For a one-year lag, the map for January shows the correlation of January and January a year later. And so forth.

All calculated are maps shown for each month. Observations of sea ice concentration are from HadISST1.1 (Rayner et al, 2003).

3.17.1 Version & Contact info

- Version 1 (1/26/2021)
- PI (Cecilia Bitz, University of Washington, bitz@uw.edu)
- Developer/point of contact (Cecilia Bitz, University of Washington, bitz@uw.edu)
- Other contributors: Lettie Roach

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

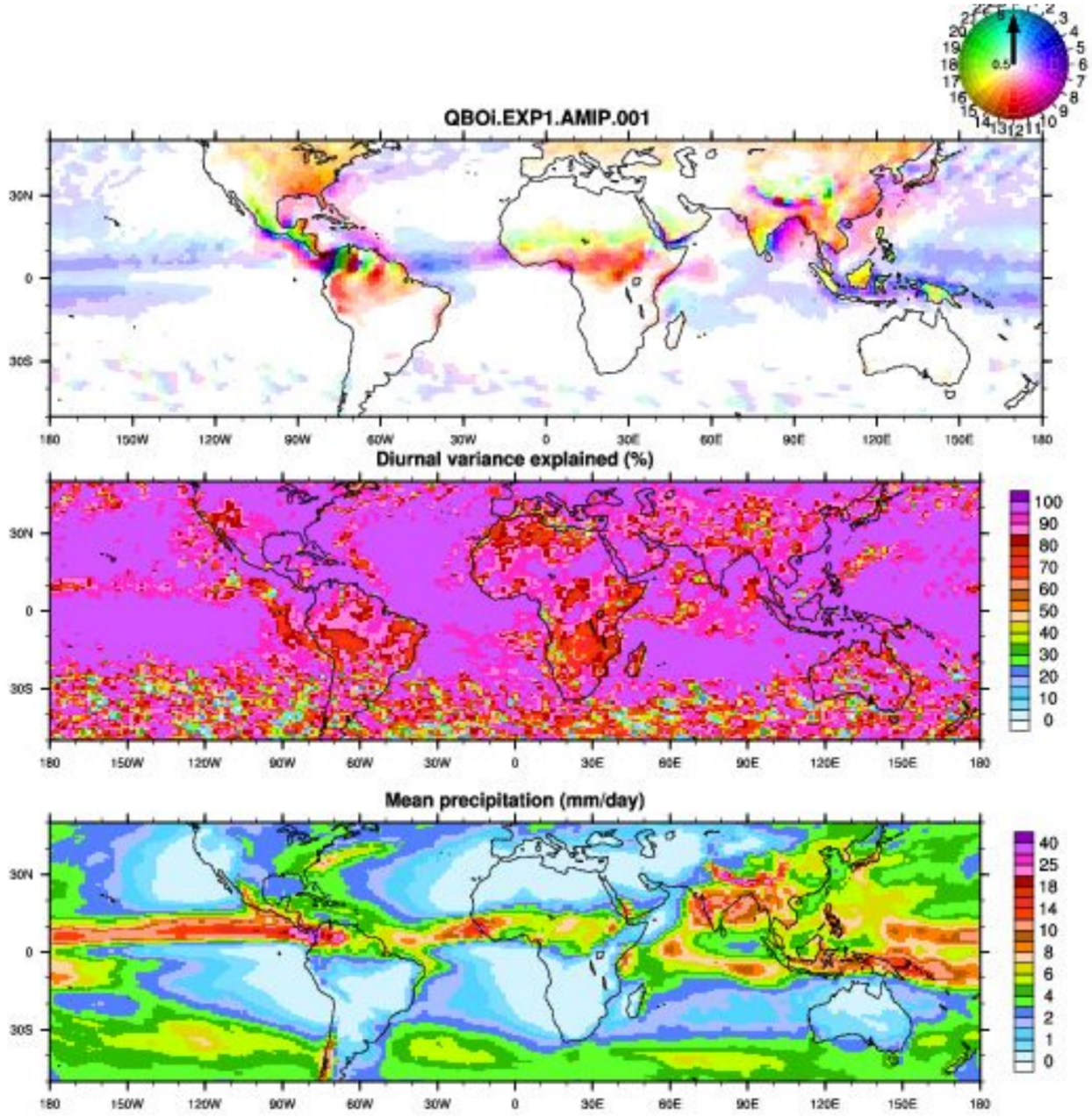


Fig. 8: Figure 1: Diurnal precipitation analysis for the tropics in June/July/August for output from a default CESM case, showing (a) the local timing maximum of the peak in the first harmonic of diurnal rainfall, (b) the variance explained by the first harmonic of the diurnal rainfall variability (%), and the mean precipitation over the analysis period (mm/day).

3.17.2 Functionality

The current package consists of two files:

1. seaice_suite_sic_mean_sigma.py which loads data, calculates statistics, regrids to grid of observations and plots everything.
2. seaice_MLD_stats.py contains functions for computing the linear regression and lagged correlation used in this POD and also our mixed-layer depth POD.

3.17.3 Required programming language and libraries

Python version 3, numpy, pandas, scipy, netCDF4, cftime, xarray, dask, esmpy, xesmf, matplotlib, cartopy

3.17.4 Required model output variables

Monthly mean sea ice concentration on the original model grid for 1979-2014 from a historical simulation.

3.17.5 References

1. Roach, L., C.M. Bitz, et al. In preparation.
2. Rayner, N. A.; Parker, D. E.; Horton, E. B.; Folland, C. K.; Alexander, L. V.; Rowell, D. P.; Kent, E. C.; Kaplan, A. (2003) Global analyses of sea surface temperature, sea ice, and night marine air temperature since the late nineteenth century J. Geophys. Res.Vol. 108, No. D14, 4407 10.1029/2002JD002670 (pdf ~9Mb)

3.18 Soil moisture-Evapotranspiration Coupling Diagnostic Package

Last update: 6/28/2019

The Soil moisture-Evapotranspiration (SM-ET) Coupling Diagnostic Package evaluates the relationship between SM and ET in the summertime. It computes the correlation between surface (top-10cm) SM and ET, at the interannual timescale, using summertime-mean values. Positive correlation values indicate that, at the interannual time scale (from one summer to the next), soil moisture variability controls ET variability. This can generally be expected to occur when soil moisture availability is the limiting factor for ET. Conversely, negative values indicate that ET variations drive variations in soil moisture levels, which can be expected to occur in regions where soil moisture is plentiful and the limiting factor for ET becomes atmospheric evaporative demand (radiation, temperature); it also reflects the anticorrelation between precipitation, which drives soil moisture, and radiation, which drives ET. In addition to its sign, the correlation value quantifies how much of ET interannual variability is explained by soil moisture variations (if the correlation is positive; vice versa if it is negative)—in other words, the tightness of the SM–ET relationship. Considering seasonal means removes issues associated with the coseasonality of soil moisture and ET, while still reflecting the overall (i.e., seasonally integrated) dependence of ET on soil moisture throughout the whole season. See Berg and Sheffield (2018) for further details.

3.18.1 Contact info

- PIs of the project: Eric Wood, Princeton University (efwood@princeton.edu);
- previous PI Justin Sheffield, formerly at Princeton University, now at University of Southampton, UK (justin.sheffield@soton.ac.uk).
- Current developer: Alexis Berg (ab5@princeton.edu)

Open source copyright agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.18.2 Functionality

The currently package consists of:

- a Python script (SM_ET_coupling.py), which sets up the directories and calls...
- ... an R script (SM_ET_coupling.R) which reads the data, performs the calculations and generates the plots.

As a module of the MDTF code package, all scripts of this package can be found under `mdtf/MDTF_$ver/var_code/SM_ET_coupling` and pre-digested observational data (in RData format) under `mdtf/inputdata/obs_data/SM_ET_coupling` Place your input data at: `mdtf/inputdata/model/$model_name/mon/`

3.18.3 Required programming language and libraries

Python and R are required to run the diagnostic.

The part of the package written in Python requires packages os and subprocess. These Python packages are already included in the standard Anaconda installation The R script requires packages ColorRamps, maps, fields, akima and ncdf4. R version 3.4 was used to develop this package, but it should work on older and more recent R versions.

3.18.4 Required model output variables

The following three 3-D (lat-lon-time), monthly model fields are required:

- surface soil moisture ("mrsos" in CMIP5 conventions)
- land evaporation ("evspsbl") or latent heat flux ("hfls")
- precipitation ("pr")

The observational estimate from GLEAM (see below) is for 1980-2014; therefore, the model data should cover the same time period, as the background climate, and thus the SM-ET coupling, could be different if the model data covers another period (although we attempt to control for precipitation differences between model and observations – see below). Note that 2014 is the end year of the historical period of CMIP6 historical simulations. 1980 is the beginning of the GLEAM data. Note that, by default, the R script will read the whole monthly model file provided as input. We thus recommend that users truncate their model files to cover precisely the period 1980-2014.

3.18.5 More about the diagnostic

a) Choice of reference dataset

With respect to SM-ET coupling, an observational value of the metric is difficult to obtain, because of the challenges associated with measuring soil moisture and evapotranspiration extensively over continents, at the required spatial and temporal scales. Global observational products of ET and soil moisture do exist (in particular from remote sensing), but are plagued by numerous uncertainties or shortcomings limiting their use, here, to compute SM-ET coupling in a straightforward manner. Calculating SM-ET coupling from various datasets combining modeling and observations, such as reanalyses (e.g., MERRA2, JRA55, ERA-I) or land surface models driven by observations (e.g., GLADS, GLDAS2), yields estimates of SM-ET coupling that exhibit significant spread (comparable in some regions to the spread across CMIP5 models), even though their representation of the driving surface climate (e.g., precipitation) is very comparable. This diversity is not necessarily surprising, given that SM-ET coupling largely remains, in these types of products, a product of the underlying land model used to create the dataset. In this context, we eventually decided to use the GLEAM (Global Land Evaporation Amsterdam Model) dataset (*Martens et al. 2017* (page 119); see https://www.gleam.eu/), as a reference, provided along the SM-ET coupling metric here in the diagnostic package. GLEAM is a global, gridded land surface dataset based on remote sensing covering 1980-2017 (here we only use 1980-2014, so that, in particular, the latest CMIP6 simulations, which extend to 2014, can be compared to this data). While the control of SM on ET in the GLEAM dataset ultimately remains a property of the modeling assumptions underlying this product, GLEAM is the only product, to our knowledge, assimilating available remote sensing observations of both soil moisture and vegetation, thus providing a dataset including both observationally-constrained and mutually-consistent SM and ET. That being said, caution should be exerted when comparing the model results to this estimate, which is only provided as a tentative reference, not an observational truth.

b) Correction for precipitation differences

In *Berg and Sheffield (2018)* (page 119), we found that across CMIP5 models, differences in summertime precipitation explained a significant part of model differences in SMET coupling. In other words, in a given location – for instance, a semi-arid location models with more precipitation have less positive SM-ET coupling – i.e., ET is less limited by soil moisture (see Figure 3 in *Berg and Sheffield 2018* (page 119)). However, mean precipitation did not explain all of the differences across models, which we interpreted as reflecting model differences, for a given amount of precipitation, in the treatment of land surface processes related to vegetation and hydrology. In the diagnostic package here, summertime precipitation differences between the model and the observations (GLEAM over 1980-2014) are provided as a plot. Assuming that, to first approximation, precipitation differences are independent from the surface, we attempt to control for precipitation differences between model and observations in the package by using the regression across CMIP5 models between mean summertime precipitation and SM-ET coupling established in Berg and Sheffield (*2018* (page 119); Figure 3). In other words, the coupling calculated for the model, when correcting for precipitation differences, is the coupling that would have existed in the model if precipitation were correct (i.e., equal to the observations in GLEAM). For instance, in regions where the model produces too much rainfall, the correction will tend to increase the estimate of SM-ET coupling (since, if precipitation was more realistic, it would be lower and soil moisture control on ET would thus be greater). This correction is tentative, as it assumes that the relationship across CMIP5 models between precipitation and SM-ET coupling is realistic, in the sense that it says something about the physics of the real world.

3.18.6 References

1. Berg A. and J. Sheffield (2018), Soil moisture-evapotranspiration coupling in CMIP5 models: relationship with simulated climate and projections, *Journal of Climate*, **31** (12), 4865-4878.
2. Martens, B., Miralles, D.G., Lievens, H., van der Schalie, R., de Jeu, R.A.M., FernándezPrieto, D., Beck, H.E., Dorigo, W.A., and Verhoest, N.E.C.: GLEAM v3: satellite-based land evaporation and root-zone soil moisture, *Geoscientific Model Development*, **10**, 1903–1925, https://doi.org/10.5194/gmd-10-1903-2017, 2017.

3.19 Stratosphere-Troposphere Coupling: Annular Modes

Last update: 2023-03-28

This POD assesses characteristics of the annular modes as a function of day of year. It makes four kinds of figures from provided model data:

1. EOF1 pattern plots on standard pressure levels, representing the annular mode structures throughout the troposphere and stratosphere. (cf., Gerber et al. 2010, Fig. 4; Simpson et al., 2011, Fig. 1)
2. Annular mode interannual standard deviation (cf., Gerber et al., 2010, Fig. 7)
3. Annular mode e-folding timescales (or "persistence"; cf., Gerber et al. 2010, Fig. 8; Kidston et al., 2015, Fig. 1)
4. Annular mode predictability (cf., Gerber et al., 2010, Fig. 9)

All figures are made for both hemispheres. This POD also outputs the computed annular mode indices and EOF structures as netcdf files.

The figure from (1) should always be viewed to verify the spatial structure of the annular modes; if the spatial patterns for model data do not match the observations (or the figures in the papers referenced above), then the 1st EOF may not represent an "annular mode"-like pattern, which means caution is warranted for interpreting the other figures and the digested output data.

The figures from (2) highlight the seasonal cycle in annular mode variability. The figures from (3) show estimates of the seasonally varying persistence of the annular modes. Lastly, the figures from (4) demonstrate what fraction of the variance of the annular mode at a given pressure level (default 850 hPa) can be "predicted" using a persistence forecast of the annular mode at other levels (see Gerber et al., 2010 for full details).

3.19.1 Version & Contact info

- Version/revision information: v1.0 (Mar 2023)
- Project PIs: Amy H. Butler (NOAA CSL) and Zachary D. Lawrence (CIRES / NOAA PSL)
- Developer/point of contact: Zachary Lawrence (zachary.lawrence@noaa.gov)

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

3.19.2 Functionality

This POD is composed of three files, including the main driver script `stc_annular_modes.py`, the functions that perform the diagnostic computations in `stc_annular_modes_calc.py`, and the functions that compile the specific POD plots in `stc_annular_modes_plot.py`. The driver script reads in the necessary data, calls the computation functions to perform the anomaly calculations, the EOF analysis, and annular mode diagnostics.

The observational data this POD uses is based on ERA5 reanalysis (Hersbach, et al., 2020) zonal mean geopotential heights.

By default, the annular modes are computed for input model data as described in Gerber et al., 2010; in short, for every timestep, the global mean geopotential height is removed, and then a 60 day low-pass filter is applied across the daily data before a 30 year low-pass filter is applied across the days of year. This process is intended to remove a slowly varying climatology that helps to remove trends, such that the anomalies represent "true" variations. The annular modes are then assumed to be the 1st EOF of these daily anomalies between 20-90 degrees latitude (for the Northern and Southern hemispheres).

Note: Users can opt to adjust this POD's settings.jsonc file to instead compute the annular modes using a "simple" method, which computes anomalies by removing the global mean heights, removing a standard climatology, and linearly detrending the anomalies across the days of year. However, the pre-digested observational data provided with this POD are computed using the "gerber" method.

3.19.3 Required programming language and libraries

This POD requires Python 3, with the following packages:

- numpy
- scipy
- xarray
- pandas
- eofs
- matplotlib

3.19.4 Required model output variables

Only one daily mean field on pressure levels is required:

- Zonal Mean Geopotential Height, `zg` as `(time,lev,lat)` (units: m)

Ideally, this data should span pressure levels between 1000 and 1 hPa. Results will be plotted for this range of levels. However, absent/missing data will properly have blanks in the output figures.

3.19.5 Scientific background

The Northern and Southern Annular Modes (NAM/SAM) are the dominant large-scale circulation variability patterns of the extratropics (Thompson and Wallace, 2000). They represent fluctuations of mass into or out of the polar cap regions, manifesting as patterns of similarly signed height/pressure anomalies in the polar cap surrounded by an opposite signed ring of anomalies in the midlatitudes (hence their "annular" appearance).

The annular modes also characterize a coupled pattern of variability between the troposphere and stratosphere (Gerber et al., 2010; Kushnir 2010; Simpson et al., 2011). In the troposphere, the NAM/SAM roughly correspond to the strength and latitudinal position of the mid-latitude jets; in the wintertime stratosphere, they represent the strength of the stratospheric polar vortex. During these winter periods, the state of the stratosphere can have a "downward influence" on the tropospheric annular mode state, whereby anomalies in the strength of the stratospheric vortex can drive persistent same-signed tropospheric annular mode phases (Baldwin and Dunkerton 2001). The resultant influence on the position of the jets can further impact regional shifts in large-scale weather patterns. Thus, while the tropospheric annular modes can evolve year-round, the stratosphere-troposphere coupling that occurs during winter/spring drives a distinct seasonal cycle in tropospheric annular mode variance and persistence (Gerber et al., 2010; Simpson et al. 2011; Schenzinger & Osprey 2015).

In the Northern Hemisphere, the aforementioned sort of "downward influence" is generally organized around midwinter extreme vortex events such as sudden stratospheric warmings and vortex intensifications. However, in the Southern Hemisphere, stratosphere-troposphere annular mode coupling is typically organized around the seasonal breakdown of the polar vortex in late spring. As a result, the seasonal cycle in annular mode variability and persistence tends to occur close in time in both hemispheres, maximizing around December-February in the Northern Hemisphere, and October-December in the Southern Hemisphere (Kidston et al., 2015).

A misrepresentation of stratospheric variability in models can lead to biases in annular mode coupling (Gerber et al., 2010; Simpson et al., 2011). For instance, a lack of SSWs or too late final warmings can shift the seasonal cycle in annular mode variance/persistence too late in models relative to observations. Processes that affect stratospheric polar vortex variability in models (e.g., model lid height, gravity wave parameterizations, interactive chemistry, etc.), can thus potentially affect the representation of the tropospheric jets, regional weather, and the statistics of temperatures and precipitation through the annular mode "pathway". However, it is also possible for model biases in the annular modes to arise separately from the stratosphere due to poorly represented processes such as low-level orographic drag (Pithan et al., 2016).

3.19.6 References

Thompson, D. W. J., and J. M. Wallace, 2000: Annular Modes in the Extratropical Circulation. Part I: Month-to-Month Variability. J. Climate, 13, 1000–1016, https://doi.org/10.1175/1520-0442(2000)013<1000:AMITEC>2.0.CO;2.

Baldwin, M. P., and T. J. Dunkerton, 2001: Stratospheric harbingers of anomalous weather regimes. Science, 294(5542), 581-584, https://doi.org/10.1126/science.1063315

Baldwin, M.P. and D.W.J. Thompson, 2009: A critical comparison of stratosphere–troposphere coupling indices. Q.J.R. Meteorol. Soc., 135: 1661-1672, https://doi.org/10.1002/qj.479

Gerber, E. P., et al. 2010: Stratosphere-troposphere coupling and annular mode variability in chemistry-climate models, J. Geophys. Res., 115, D00M06, https://doi.org/10.1029/2009JD013770.

Kushner, P. J., 2010: Annular modes of the troposphere and stratosphere. The Stratosphere: Dynamics, Transport, and Chemistry, 190, 59-91., https://doi.org/10.1029/GM190

Simpson, I. R., P. Hitchcock, T. G. Shepherd, and J. F. Scinocca, 2011: Stratospheric variability and tropospheric annular-mode timescales, Geophys. Res. Lett., 38, L20806, https://doi.org/10.1029/2011GL049304.

Kidston, J., et al. 2015: Stratospheric influence on tropospheric jet streams, storm tracks and surface weather. Nature Geosci 8, 433–440, https://doi.org/10.1038/ngeo2424

Schenzinger, V., and S. M. Osprey, 2015: Interpreting the nature of Northern and Southern Annular Mode variability in CMIP5 Models, J. Geophys. Res. Atmos., 120, 11,203– 11,214, https://doi.org/10.1002/2014JD022989.

Pithan, F., T. G. Shepherd, G. Zappa, and I. Sandu 2016: Climate model biases in jet streams, blocking and storm tracks resulting from missing orographic drag, Geophys. Res. Lett., 43, 7231–7240, https://doi.org/10.1002/2016GL069551.

Hersbach, H. and coauthors, 2020: The ERA5 global reanalysis. Q J R Meteorol Soc., 146, 1999-2049, https://doi.org/10.1002/qj.3803

3.20 Stratosphere-Troposphere Coupling: Eddy Heat Fluxes

Last update: 2022-09-26

This POD assesses the interaction of vertically propagating planetary-scale stationary waves on the polar winter/spring stratosphere. The vertical component of the Eliassen-Palm Flux is approximately proportional to the zonal mean eddy heat flux, v'T', where v is the meridional wind, and T is the temperature (see Andrews et al., 1987). Thus, this POD uses the eddy heat flux at 100 hPa as a proxy for the vertical flux of waves entering the stratosphere.

In general, when the time-integrated eddy heat flux in the lowermost stratosphere is above normal, the polar stratospheric circulation should be weaker than normal with warmer temperatures; similarly, when the eddy heat flux is below normal, the circulation should be stronger than normal with colder temperatures (see Newman, et al., 2001). The eddy heat fluxes entering the stratosphere are primarily driven by vertically propagating planetary-scale Rossby waves which have both stationary and transient components. This POD calculates eddy heat fluxes using monthly mean fields, and thus it primarily measures these relationships for stationary waves (since in the monthly mean the transient waves will be averaged out).

This POD makes two kinds of figures from provided model data:

- Scatterplots of early-season eddy heat fluxes with late-season polar cap temperatures
- Lag-correlation plots of polar cap geopotential heights at different pressure levels and months with early-season eddy heat fluxes at 100 hPa

These plots are made for both hemispheres; for the Northern Hemisphere (NH), they focus on the early (DJF or Dec) and late winter (JFM). For the Southern Hemisphere (SH), they focus on the early (ASO or Sep) and late spring (SON). These months are when coupling between the stratosphere and troposphere are most active in the respective hemispheres.

Polar stratospheric circulation variability is known to influence tropospheric weather and climate (see Kidston et al., 2015). Different teleconnections, like those related to ENSO, are sometimes considered to have stratospheric pathways through which they act. These stratospheric teleconnection pathways are generally related to how a given phenomenon influences stratospheric circulation variability, and the subsequent coupling of the stratospheric state with the troposphere.

In a simple sense, this POD evaluates the "first step" of stratosphere-troposphere coupling – that is, the tropospheric influence on driving stratospheric circulation anomalies. If a model underestimates or misrepresents this "upward coupling", they can further miss or underestimate the impact of "downward coupling" related to the stratosphere. Issues in modeling these processes can be related to model characteristics such as vertical resolution, the height of the model lid, and the representation of parameterized processes.

3.20.1 Version & Contact info

- Version/revision information: v1.0 (Jun 2022)
- Project PIs: Amy H. Butler (NOAA CSL) and Zachary D. Lawrence (CIRES / NOAA PSL)
- Developer/point of contact: Zachary Lawrence (zachary.lawrence@noaa.gov)

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

3.20.2 Functionality

The entirety of this POD is contained in the file `stc_eddy_heat_fluxes.py`. This script reads in the model fields, calculates zonal mean eddy heat fluxes and polar cap temperatures and geopotential heights, and generates the plots.

The observational data this POD uses is based on ERA5 reanalysis (Hersbach, et al., 2020), and includes pre-computed zonal mean eddy heat fluxes, temperatures, and geopotential heights (i.e., they have dimensions of `(time,level, lat)`) calculated from monthly mean fields.

3.20.3 Required programming language and libraries

This POD requires Python 3, with the following packages:

- numpy
- scipy
- xarray
- xesmf
- matplotlib

3.20.4 Required model output variables

The following monthly mean fields are required:

- Temperature at 50 hPa, `t50` as `(time,lat,lon)` (units: K)
- Temperature at 100 hPa, `t100` as `(time,lat,lon)` (units: K)
- Meridional Winds at 100 hPa, `v100` as `(time,lat,lon)` (units: m/s)
- Geopotential Height, `zg` as `(time,level,lat,lon)` (units: m)

3.20.5 References

Andrews, D. G., J. R. Holton, and C. B. Leovy, 1987: Middle Atmosphere Dynamics, Academic press, No. 40.

Furtado, J. C., J. L. Cohen, A. H. Butler, E. E. Riddle, and A. Kumar, 2015: Eurasian snow cover variability and links to winter climate in the CMIP5 models. Clim Dyn, 45, 2591–2605, https://doi.org/10.1007/s00382-015-2494-4.

Hersbach, H. and coauthors, 2020: The ERA5 global reanalysis. Q J R Meteorol Soc., 146, 1999-2049, https://doi.org/10.1002/qj.3803

Kidston, J., A. Scaife, S. C. Hardiman, D. M. Mitchell, N. Butchart, M. P. Baldwin, and L. J. Gray, 2015: Stratospheric influence on tropospheric jet streams, storm tracks and surface weather. Nature Geosci 8, 433–440. https://doi.org/10.1038/ngeo2424

Newman, P. A., E. R. Nash, and J. E. Rosenfield, 2001: What controls the temperature of the Arctic stratosphere during the spring? JGR: A, 106, 19999–20010, https://doi.org/10.1029/2000JD000061.

3.20.6 More about this POD

Sign of eddy heat fluxes in NH vs SH

In the Northern Hemisphere (NH), positive eddy heat fluxes represent poleward and upward wave fluxes. However, in the Southern Hemisphere (SH), the sign is flipped such that negative eddy heat fluxes represent the poleward and upward wave fluxes. This means that the statistical relationships evaluated in this POD will generally be opposite-signed for the SH figures.

Use of bootstrapping

The scatterplots provided by this POD show the correlations between the 100 hPa eddy heat flux and 50 hPa polar carp temperatures. In these figures, the parentheses next to the correlations contain the 95% bootstrap confidence interval on the correlations from resampling the available years 1000 times. These confidence intervals help to determine whether the correlations are significant; if 0 does not fall within the range of the confidence interval, the correlation can be said to be statistically significant. Furthermore, the bootstrap confidence interval in the observation plots give a sense of the sampling variability in the historical record; if the model correlation falls outside the observed bootstrap confidence interval, it is fair to say the model has a too strong or too weak relationship.

3.21 Stratosphere-Troposphere Coupling: Stratospheric Ozone and Circulation

Last update: 2023-01-31

This POD assesses coupling between stratospheric ozone and the large-scale circulation. Ozone-circulation coupling occurs during spring when sunlight returns to the polar region and the radiative influence of ozone anomalies drives changes to meridional temperature gradients and thus zonal winds, which can then dynamically drive temperatures changes, which feedback onto ozone chemistry. For example, in years when the Antarctic ozone hole is larger (more ozone loss) in early spring, the polar vortex stays stronger and persists later, leading to a later transition of the vortex at 50 mb to its summertime state, here defined zonal-mean zonal winds at 60 degLat as less than 5 (15) m/s in the NH (SH). This seasonal transition of the polar vortex is called the "final stratospheric warming". Because the Arctic rarely gets cold enough for severe chemical ozone loss, ozone-circulation coupling is primarily observed in the Southern Hemisphere, but this POD allows application to both hemispheres, as similar relationships may still occur in the Northern Hemisphere during extreme polar conditions.

This POD makes four kinds of figures from provided model data:

- Scatterplots of early-spring polar cap stratospheric ozone with late-spring zonal winds
- Scatterplots of early-spring polar cap stratospheric ozone with final stratospheric warming day of year
- Lag-correlation plots of polar cap stratospheric ozone with extratropical zonal winds for different pressure levels
- Linear trends of polar cap ozone, temperature, and extratropical zonal winds as a function of month and pressure level

These plots are made for both hemispheres, with a focus on spring. This season is when sunlight returns to the polar region and when the strongest coupling between stratospheric ozone and the circulation appears. The metrics used are designed to focus on processes with known biases, particularly in the Southern Hemisphere. For example, the scatterplots showing late-spring zonal winds or final stratospheric warming day of year can be used to compare the mean values of these quantities in the model with reanalysis. In the SH, CMIP models tend to have too late of final warming, or equivalently, too strong of late spring polar vortex winds (Wilcox et al., 2013). The POD outputs some of these metrics so that multi-model comparison can be performed.

Note that many CMIP6 models do not have interactive stratosheric chemistry, and instead use prescribed ozone provided by Checa-Garcia et al. (2018a,b), except for three models that instead use prescribed ozone from simulations performed by the CESM-WACCM model (CESM2, CESM2-FV2, NorESM2). Details of ozone in CMIP6 models can be found in Keeble et al. (2021). In models with prescribed ozone, the ozone forcing will still influence the circulation, but the circulation changes cannot feedback onto ozone, which may influence the degree to which they capture the full response in both hemispheres (Haase et al., 2020, Friedel et al. 2022).

3.21.1 Version & Contact info

- Version/revision information: v1.0 (Jan 2023)
- Project PIs: Amy H. Butler (NOAA CSL) and Zachary D. Lawrence (CIRES/NOAA PSL)
- Developer/point of contact: Amy Butler (amy.butler@noaa.gov)

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

3.21.2 Functionality

This POD is driven by the file `stc_ozone.py`, with a helper script of `stc_ozone_defs.py`. The driver script reads in the model fields, calculates zonal mean zonal winds for user-defined latitude bands, and polar cap ozone and temperature, and generates the plots. It also estimates the final warming DOY using monthly-mean zonal wind data at 60 degLat, as in Hardimann et al. (2011).

The observational data this POD uses is based on ERA5 reanalysis (Hersbach, et al., 2020), and includes pre-computed zonal mean zonal winds, temperatures, and ozone (i.e., they have dimensions of `(time,lev,lat)`) calculated from monthly mean fields.

3.21.3 Required programming language and libraries

This POD requires Python 3, with the following packages:

- numpy
- datetime
- scipy
- xarray
- matplotlib

3.21.4 Required model output variables

The following monthly mean fields are required:

- Temperature, `ta` as `(time,lev,lat,lon)` (units: K)
- Zonal Winds, `ua` as `(time,lev,lat,lon)` (units: m/s)
- Ozone, `o3` as `(time,lev,lat,lon)` (units: mol mol-1)

3.21.5 References

Hardiman, S. C., et al., 2011: Improved predictability of the troposphere using stratospheric final warmings, J. Geophys. Res., 116, D18113, doi:10.1029/2011JD015914

Hersbach, H. and coauthors, 2020: The ERA5 global reanalysis. Q J R Meteorol Soc., 146, 1999-2049, https://doi.org/10.1002/qj.3803

Checa-Garcia, R: CMIP6 Ozone forcing dataset, 2018: supporting information, Zenodo, https://doi.org/10.5281/zenodo.1135127

Checa-Garcia, R., Hegglin, M. I., Kinnison, D., Plummer, D. A., and Shine, K. P., 2018: Historical Tropospheric and Stratospheric Ozone Radiative Forcing Using the CMIP6 Database, Geophys. Res. Lett., 45, 3264–3273, https://doi.org/10.1002/2017GL076770

Keeble, J., Hassler, B., Banerjee, A., Checa-Garcia, R., Chiodo, G., Davis, S., Eyring, V., Griffiths, P. T., Morgenstern, O., Nowack, P., Zeng, G., Zhang, J., Bodeker, G., Burrows, S., Cameron-Smith, P., Cugnet, D., Danek, C., Deushi, M., Horowitz, L. W., Kubin, A., Li, L., Lohmann, G., Michou, M., Mills, M. J., Nabat, P., Olivié, D., Park, S., Seland, Ø., Stoll, J., Wieners, K.-H., and Wu, T.. 2021: Evaluating stratospheric ozone and water vapour changes in CMIP6 models from 1850 to 2100, Atmos. Chem. Phys., 21, 5015–5061, https://doi.org/10.5194/acp-21-5015-2021

Haase, S., Fricke, J., Kruschke, T., Wahl, S., and Matthes, K., 2020: Sensitivity of the Southern Hemisphere circumpolar jet response to Antarctic ozone depletion: prescribed versus interactive chemistry, Atmos. Chem. Phys., 20, 14043–14061, https://doi.org/10.5194/acp-20-14043-2020

Friedel, M., Chiodo, G., Stenke, A. et al., 2022: Springtime arctic ozone depletion forces northern hemisphere climate anomalies. Nat. Geosci. 15, 541–547, https://doi.org/10.1038/s41561-022-00974-7

Wilcox, L. J., and Charlton-Perez, A. J., 2013: Final warming of the Southern Hemisphere polar vortex in high- and low-top CMIP5 models, J. Geophys. Res. Atmos., 118, 2535– 2546, doi:10.1002/jgrd.50254

3.21.6 More about this POD

Statistical testing for correlations

One of the outputs of this POD is lag correlations between spring ozone at 50 mb and zonal-mean zonal winds at all other pressure levels for two months before and after. A student's 2-tailed t-test of the Pearson's correlation coefficient is used to determine where the correlation is significant at p<0.05. Stippling is shown where the correlations are *not* significant.

Use of bootstrapping

The scatterplots provided by this POD show the correlations between springtime ozone at 50 mb and either the final stratospheric warming day of year, or the late summer zonal winds at 50 mb. In these figures, the parentheses next to the correlations contain the 95% bootstrap confidence interval on the correlations from resampling the available years 1000 times. These confidence intervals help to determine whether the correlations are significant; if 0 does not fall within the range of the confidence interval, the correlation can be said to be statistically significant. Furthermore, the bootstrap confidence interval in the observation plots give a sense of the sampling variability in the historical record; if the model correlation falls outside the observed bootstrap confidence interval, it is fair to say the model has a too strong or too weak relationship.

Statistical testing for linear trends This POD outputs linear least squares best-fit trends in temperatures, winds, and ozone averaged for different regions in the extratropics, for two different historical periods during which ozone depletion or recovery occurred. These are calculated using the scipy function "linregress", which allows output of the p-value which is defined as: "The p-value for a hypothesis test whose null hypothesis is that the slope is zero, using Wald Test with t-distribution of the test statistic." Stippling is shown where the trends are *not* significant.

3.22 Stratosphere-Troposphere Coupling: QBO and ENSO stratospheric teleconnections

Last update: 2023-10-03

This script and its helper scripts ("stc_qbo_enso_plottingcodeqbo.py" and "stc_qbo_enso_plottingcodeenso.py") do calculations to assess the representation of stratospheric telconnections associated with the Quasi-Biennial Oscillation (QBO) and the El Nino Southern Oscillation (ENSO). This POD uses monthly 4D (time x plev x lat x lon) zonal wind, 4D meridional wind, 4D temperature, 3D (time x lat x lon) sea level pressure, and 3D sea surface temperature data. Coupling between the QBO and the boreal polar stratosphere takes place during boreal fall and winter whereas coupling between the QBO and the austral polar stratosphere takes place mainly during austral spring and summer. By default, the POD defines the QBO for NH (SH) analyses using the Oct-Nov (Jul-Aug) 5S-5N 30 hPa zonal winds. The QBO is thought to influence the polar stratospheres, the so-called "polar route," by modulating the lower stratospheric (~100-50 hPa) and middle stratospheric (~20-5 hPa) mid-latitude circulation. The aforementioned lower stratospheric teleconnection is also associated with a change in the strength and position of the tropospheric jet; the so-called "subtropical route." In addition, evidence continues to show that the QBO directly influences the underlying tropical tropospheric circulation, referred to as the "tropical route." These three teleconnections allow the QBO to elicit surface impacts globally. Said teleconnections are visualized herein by using a metric of planetary wave propagation (eddy heat flux), circulation response (zonal wind), and surface impact (sea level pressure). Additionally, metrics of model QBOs (e.g., amplitude, height, width) are produced. ENSO's coupling with the polar stratospheres takes place as the amplitude of ENSO maximizes during boreal fall and winter. By default, the POD defines ENSO for NH (SH) analyses using the Nov-Mar (Sep-Jan) Nino3.4 SSTs. Though ENSO's teleconnections are global, it interacts with the stratosphere by stimulating tropical-extratropical Rossby waves that constructively interfere with the climatological extratropical stationary wave mainly over the Pacific, promoting enhanced upward planetary wave propagation into the stratosphere. Similar to the QBO code, ENSO's teleconnections are visualized using the eddy heat flux, the zonal wind, and the sea level pressure.

This POD makes six kinds of figures and one text file from provided model data:

- Zonal-mean zonal wind anomalies (deviations from seasonal cycle) composited based on El Nino and La Nina years are shown in red/blue shading. Nina minus Nino differences are shown in shading as well and climatological winds are overlaid on all aforementioned plots in black contours
- Zonal-mean eddy heat flux anomalies composited based on El Nino and La Nina years are shown in red/blue shading. Nina minus Nino differences are shown in shading as well and climatological heat flux is overlaid on all aforementioned plots in black contours
- Sea level pressure anomalies composited based on El Nino and La Nina years are shown in red/blue shading. Nina minus Nino differences are shown in shading as well and climatological sea level pressure is overlaid on all aforementioned plots in black contours
- A text file of QBO metrics (min/mean/max QBO periodicity, easterly/westerly/total amplitude, lowest altitude tropical stratospheric isobar that the QBO reaches, the height or vertical extent of the QBO, and its latitudinal width) is produced.
- Should the above QBO metrics code detect a QBO in the model data, similar plots as the aforementioned three ENSO plots, but composited based on easterly and westerly QBO years, are made

These plots are made for both hemispheres, with a focus on winter and spring, the seasons when upward propagating planetary waves couple the troposphere and stratosphere. The metrics are designed to reveal the extratropical circulation response to two forms of tropical internal variability, which are generally difficult to represent spontaneously in climate models (QBO+ENSO). Though ENSO's representation in climate models as well as the representation of its teleconnections has significantly improved over multiple generations of CMIP experiments (Bellenger et al. 2014; Planton et al. 2021), it is less clear how ENSO's coupling with the polar stratosphere is represented by models. The sea level pressure ENSO responses reveal the precursor tropospheric forcing patterns (e.g., Aleutian Low response) that should stimulate or reduce upward planetary wave propagation into the stratosphere. The zonal wind and eddy heat flux plots reveal if the reinforced or suppressed upward planetary wave propagation due to ENSO is actually "felt" in the stratosphere and the sea level pressure plots can again be referenced for evidence of a downward propagating winter/spring annular mode response to ENSO modulating the polar vortex.

Similar plots to the ones made based on El Nino and La Nina years are made for easterly and westerly QBO years if and when a QBO is detected in the model data; e.g., models with too-coarse vertical resolution may not simulate a QBO (Zhao et al. 2018). It should be interesting to compare the QBO metrics with the representation of QBO teleconnections in models. Models struggle to represent several QBO attributes (Richter et al. 2020) and since the structure of the QBO (e.g., its amplitude or latitudinal width) is intimately tied to the representation of QBO teleconnections (Garfinkel and Hartmann 2011; Hansen et al. 2013), models generally have a difficult time representing the extratropical impacts of the QBO (Rao et al. 2020).

3.22.1 Version & Contact info

- Version/revision information: v1.0 (Oct 2023)
- Project PIs: Amy H. Butler (NOAA CSL) and Zachary D. Lawrence (CIRES/NOAA PSL)
- Developer/point of contact: Dillon Elsbury (dillon.elsbury@noaa.gov)

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

3.22.2 Functionality

This POD is driven by the file `stc_qbo_enso.py`, with two helper scripts called `stc_qbo_enso_plottingcodeenso.py` and `stc_qbo_enso_plottingcodeqbo.py`. The driver script reads in the model fields, identifies El Nino/La Nina years and easterly/westerly QBO years, computes the eddy heat flux, retrieves the QBO metrics, and then uses the two helper scripts to make the associated zonal wind, eddy heat flux, and sea level pressure plots.

The atmospheric observational data this POD uses is based on ERA5 reanalysis (Hersbach, et al., 2020), and includes pre-computed monthly zonal-mean zonal winds, zonal- mean eddy heat fluxes, and sea level pressure. The oceanic observational data that this POD uses is from HadiSST (Rayner et al. 2003) and includes pre-computed monthly sea surface temperature.

3.22.3 Required programming language and libraries

This POD requires Python 3, with the following packages:

- numpy
- xarray
- xesmf
- os
- matplotlib
- cartopy
- scipy

3.22.4 Required model output variables

The following monthly mean fields are required:

- Zonal Winds, `ua` as `(time,lev,lat,lon)` (units: m/s)
- Meridional Winds, `va` as `(time,lev,lat,lon)` (units: m/s)
- Temperature, `ta` as `(time,lev,lat,lon)` (units: K)
- Sea level pressure, `psl` as `(time,lat,lon)` (units: Pa)
- Sea surface temperature, `tos` as `(time,lat,lon)` (units: Kelvin)

3.22.5 References

Bellenger, H., Guilyardi, E., Leloup, J., Lengaigne, M., & Vialard, J. (2014). ENSO representation in climate models: From CMIP3 to CMIP5. Climate Dynamics, 42, 1999-2018, https://doi.org/10.1007/s00382-013-1783-z

Planton, Y. Y., Guilyardi, E., Wittenberg, A. T., Lee, J., Gleckler, P. J., Bayr, T., … & Voldoire, A. (2021). Evaluating climate models with the CLIVAR 2020 ENSO metrics package. Bulletin of the American Meteorological Society, 102(2), E193-E217, https://doi.org/10.1175/BAMS-D-19-0337.1

Zhao, M., Golaz, J. C., Held, I. M., Guo, H., Balaji, V., Benson, R., … & Xiang, B. (2018). The GFDL global atmosphere and land model AM4. 0/LM4. 0: 1. Simulation characteristics with prescribed SSTs. Journal of Advances in Modeling Earth Systems, 10(3), 691-734, https://doi.org/10.1002/2017MS001209

Hersbach, H. and coauthors, 2020: The ERA5 global reanalysis. Q J R Meteorol Soc., 146, 1999-2049, https://doi.org/10.1002/qj.3803

Richter, J. H., Anstey, J. A., Butchart, N., Kawatani, Y., Meehl, G. A., Osprey, S., & Simpson, I. R. (2020). Progress in simulating the quasi-biennial oscillation in CMIP models. Journal of Geophysical Research: Atmospheres, 125(8), e2019JD032362, https://doi.org/10.1029/2019JD032362

Garfinkel, C. I., & Hartmann, D. L. (2011). The influence of the quasi-biennial oscillation on the troposphere in winter in a hierarchy of models. Part I: Simplified dry GCMs. Journal of the Atmospheric Sciences, 68(6), 1273-1289, https://doi.org/10.1175/2011JAS3665.1

3.22.6 More about this POD

Statistical testing

A student's 2-tailed t-test is used to assess how likely it is that the Nina minus Nino anomalies and easterly QBO minus westerly QBO anomalies arise by chance for the zonal wind, eddy heat flux, and sea level pressure plots. A p-value <= 0.05 is used as the threshold for "statistical significance," which is denoted on the aforementioned figures in the third row using stippling.

3.23 Stratosphere-Troposphere Coupling: Stratospheric Polar Vortex Extremes

Last update: 2023-08-22

This POD assesses stratospheric polar vortex extremes, and the tropospheric circulation and surface patterns that precede and follow them. Extremes in the stratospheric polar vortex are closely linked to the tropospheric circulation and surface climate both before and after the event. The occurrence of polar stratospheric circulation extremes in the Northern Hemisphere (NH), such as sudden stratospheric warmings (SSWs) and polar vortex intensifications (VIs), are important aspects of stratospheric variability that rely on realistic representations of the stratosphere and the troposphere. Extremes in the strength of the Arctic polar stratospheric circulation are often preceded by known near-surface circulation patterns, and then subsequently followed by shifts in weather patterns (sometimes for weeks). SSWs in the Southern Hemisphere (SH) are rare (only one event in the satellite record), while VIs occur more often, but both events can have persistent impacts on SH mid-latitude weather.

The definition for SSW events used in this POD is the most commonly used one (Charlon and Polvani 2007): a reversal of the 10 hPa 60 deg latitude daily-mean climatological westerly zonal winds between November and March, which returns to westerly for at least 10 consecutive days prior to 30 April (so that final warmings are not included). SSWs are independent events if they are separated by at least 20 days of consecutive westerlies. The definition for VI events used in this POD is adapted from previous studies (Limpasuvan et al. 2005, Domeisen et al. 2020): an increase of the 10 hPa 60 deg latitude daily-mean zonal-mean zonal winds above the daily 80th percentile value calculated across

the full input data time period, which persists for at least 10 consecutive days. VIs are independent events if they are separated by at least 20 consecutive days below the 80th percentile.

Models often show a different SSW seasonality compared to reanalysis (Ayarzaguena et al. 2020), and may vary in their simulation of tropospheric circulation/surface patterns both preceding and following extreme stratospheric events (Ayarzaguena et al. 2020). Models with low model lids (>1 hPa in pressure) may show less persistent downward coupling than models with higher model lids (Charlton-Perez et al. 2013). SSWs and their precursor patterns and impacts have been heavily studied (Baldwin et al. 2021), but VIs less so (Limpasuvan et al. 2005).

This POD makes three kinds of figures from provided model data:

- Barplots showing the frequency of events by month over the input period
- Pressure versus lag contour plots of polar cap geopotential height anomalies, composited around all detected SSWs and VI events. These types of plots are sometimes referred to "dripping paint" plots in the scientific literature.
- Polar stereographic maps of surface air temperature and 500 hPa geopotential height anomalies averaged over the 30 days before and after all detected SSW and VI events.

Additionally, the POD outputs text files of the detected SSW and VI event dates in each hemisphere. These plots are made for both hemispheres, and require at least one event to be detected in order for the POD to create the figure.

3.23.1 Version & Contact info

- Version/revision information: v1.0 (Aug 2023)
- Project PIs: Amy H. Butler (NOAA CSL) and Zachary D. Lawrence (CIRES/NOAA PSL)
- Developer/point of contact: Amy Butler (amy.butler@noaa.gov)

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

3.23.2 Functionality

This POD is driven by the file `stc_spv_extremes.py`, with a helper script of `stc_spv_extremes_defs.py`. The driver script reads in the model fields, performs a few preparatory actions such as averaging the geopotential heights over the polar cap and removing the daily climatology to obtain anomalies, and selecting the 10 hPa zonal-mean zonal winds. The script then creates three plots (in both hemispheres, so 6 plots in total) and outputs to text files the SSW and VI dates.

The observational data this POD uses is based on ERA5 reanalysis (Hersbach, et al., 2020), and includes pre-computed daily-mean zonal mean zonal winds and geopotential heights (with dimensions of `(time,lev,lat)`), and gridded daily-mean 500 hPa geopotential heights and surface air temperatures (with dimensions of `(time,lat,lon)`).

3.23.3 Required programming language and libraries

This POD requires Python 3, with the following packages:

- numpy
- pandas
- datetime
- xarray
- matplotlib
- statsmodels
- cartopy
- scipy

3.23.4 Required model output variables

The following daily-mean fields are required:

- Zonal-mean zonal wind, `ua` as `(time,lev,lat)` (units: m/s)
- Zonal-mean geopotential heights, `zg` as `(time,lev,lat)` (units: m)
- Geopotential Heights at 500 hPa, `zg` as `(time,lat,lon)` (units: m)
- Surface air temperature, `tas` as `(time,lat,lon)` (units: K)

3.23.5 References

Charlton, A. J., and L. M. Polvani, 2007: A new look at stratospheric sudden warmings. Part I: Climatology and modeling benchmarks. Journal of Climate, 20, 449–469.

Limpasuvan, V., D. L. Hartmann, D. W. J. Thompson, K. Jeev, and Y. L. Yung, 2005: Stratosphere-troposphere evolution during polar vortex intensification. Journal of Geophysical Research, 110, D24101, https://doi.org/10.1029/2005JD006302.

Domeisen, D. I. V., and Coauthors, 2020: The role of the stratosphere in subseasonal to seasonal prediction Part I: Predictability of the stratosphere. Journal of Geophysical Research: Atmospheres, 125, e2019JD030920, https://doi.org/10.1029/2019JD030920.

Ayarzagüena, B., and Coauthors, 2020: Uncertainty in the Response of Sudden Stratospheric Warmings and Stratosphere-Troposphere Coupling to Quadrupled CO2 Concentrations in CMIP6 Models. Journal of Geophysical Research: Atmospheres, 125, e2019JD032345, https://doi.org/10.1029/2019JD032345.

Baldwin, M. P., and Coauthors, 2021: Sudden Stratospheric Warmings. Reviews of Geophysics, 59, e2020RG000708, https://doi.org/10.1029/2020RG000708.

Hersbach, H. and coauthors, 2020: The ERA5 global reanalysis. Q J R Meteorol Soc., 146, 1999-2049, https://doi.org/10.1002/qj.3803

3.23.6 More about this POD

Confidence intervals for frequency of events

This POD calculates the total frequency of SSW and VI events over the input period, and then determines what fraction of those events occur in each month of the winter season. Because the event either occurs or doesn't in any given month, we calculate the binomial proportion confidence interval using the Wilson score interval, for the 95% level.

Significance for vertical composites

The lag-pressure composites ("dripping paint") plots provided by this POD show the composite-mean values of standardized polar cap geopotential height anomalies. In these figures, significance is evaluated at the 95% level using a one-sample t-test, and assumes that the population mean has an anomaly value of 0 and that the sample mean comes from a normally distributed population. This may not be a robust assumption, but here this test is chosen for a computationally inexpensive estimate of significance. In these plots, values that are *insignificant* by this test are stippled.

3.24 Stratosphere-Troposphere Coupling: Vertical Wave Coupling

Last update: 2023-03-10

This POD assesses the seasonality and extremes of vertical planetary wave coupling between the extratropical troposphere and stratosphere. It makes four kinds of figures from provided model data:

1. Climatological time series of planetary wave amplitudes in the troposphere (500 hPa) and stratosphere (10 hPa)
2. NH winter and SH spring distributions of 50 hPa polar cap eddy heat fluxes (Shaw et al., 2014; Dunn-Sigouin and Shaw, 2015; England et al., 2016)
3. Composite maps of eddy geopotential heights and anomalies during extreme heat flux days (Shaw et al., 2014; England et al., 2016)
4. Correlation coherence of planetary waves between 10 and 500 hPa (Randel 1987; Shaw et al., 2014)

All figures are made for both hemispheres. The plots from (2) and (3) focus primarily on the JFM and SON periods for the NH and SH, respectively, as these are generally the seasons with the greatest variability/extremes in heat fluxes. The plots from (1) and (4) show full-season perspectives.

The figures from (1) and (2) together evaluate statistical characteristics of the planetary waves as a function of day of year and season. The figures from (3) show composite maps during extreme heat flux events relative to climatological stationary wave patterns, which help to assess the vertically-deep wave patterns associated with upward/downward propagation. The figures from (4) demonstrate the lag times at which planetary waves in the stratosphere and troposphere are most coherent, with positive lag times indicative of upward propagation and negative lag times indicative of downward propagation.

3.24.1 Version & Contact info

- Version/revision information: v1.0 (Mar 2023)
- Project PIs: Amy H. Butler (NOAA CSL) and Zachary D. Lawrence (CIRES / NOAA PSL)
- Developer/point of contact: Zachary Lawrence (zachary.lawrence@noaa.gov)

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

3.24.2 Functionality

This POD is composed of three files, including the main driver script `stc_vert_wave_coupling.py`, the functions that perform the diagnostic computations in `stc_vert_wave_coupling_calc.py`, and the functions that compile the specific POD plots in `stc_vert_wave_coupling_plot.py`. The driver script reads in the necessary data, calls the computation functions to perform Fourier decomposition, and sends the digested data to the plotting functions. The POD computes Fourier coefficients of 10 and 500 hPa geopotential heights for the 60 degrees N/S latitudes and 45-80 degree latitude bands for zonal waves 1-3. It also computes zonal wave decomposed polar cap (60-90 degrees lat) eddy heat fluxes at 50 hPa.

The observational data this POD uses is based on ERA5 reanalysis (Hersbach, et al., 2020), and includes the same diagnostics described above. The observational data also includes eddy geoopotential height fields for the NH JFM seasons, and SH SON seasons, which are used to plot composite maps during extreme heat flux events.

3.24.3 Required programming language and libraries

This POD requires Python 3, with the following packages:

- numpy
- scipy
- xarray
- pandas
- matplotlib

3.24.4 Required model output variables

The following daily mean fields are required:

- Temperature at 50 hPa, `ta50` as `(time,lat,lon)` (units: K)
- Meridional wind at 50 hPa, `va50` as `(time,lat,lon)` (units: m/s)
- Geopotential Height at 10 hPa, `zg10` as `(time,lat,lon)` (units: m)
- Geopotential Height at 500 hPa, `zg500` as `(time,lat,lon)` (units: m)

3.24.5 Scientific background

Wave motions in the polar stratosphere are primarily dominated by vertically propagating Rossby waves from the troposphere. Because of so-called "Charney-Drazin filtering", only the largest planetary scale waves are able to propagate into the stratosphere when there are westerly mean winds (Charney & Drazin, 1961; Andrews et al., 1987). As a result, vertical wave coupling between the troposphere and stratosphere follows a distinct seasonal cycle organized around the formation of the westerly stratospheric polar vortex in autumn (when waves can enter the stratosphere), and its breakdown in spring/early summer (when easterly winds prevent propagation).

The propagation characteristics of planetary waves are strongly dependent on the background mean flow, which can influence where/how these waves propagate. In some cases these can lead to events in which waves are reflected from

the stratosphere back into the troposphere, which tend to occur most often in late winter in the NH and spring in the SH (Shaw et al., 2010). These reflected waves can directly influence the tropospheric circulation.

Wave events in the stratosphere are associated with meridional fluxes of heat that can be characterized by "eddy heat fluxes" (v'T', where v is the meridional wind, and T is the temperature, and primes denote deviations from the zonal mean), which are proportional to the wave vertical group velocity under linear wave theory (Andrews et al., 1987). Statistically extreme heat flux events thus represent extremes in wave propagation with vertically deep planetary wave structures (Dunn-Sigouin and Shaw, 2015); extraordinarily high heat fluxes weaken and warm the polar vortex, whereas negative heat fluxes are generally associated with wave reflection that can dynamically cool and strengthen the vortex.

Improper representation of the wave coupling between the troposphere and stratosphere can significantly influence the tropospheric stationary wave pattern, and be tied to climatological biases in the positions of the tropospheric jets (Shaw et al., 2014b, England et al., 2016). Biases in the stratospheric circulation that can arise from, e.g., too little parameterized gravity wave drag can also affect how planetary waves propagate in the stratosphere and affect the occurrence of extreme heat flux events. Model characteristics such as the height of the model top, and the implementation (or lack of) sponge layers near the model top can additionally lead to unphysical excessive damping or reflection of waves, which can subsequently influence biases in the tropospheric stationary wave patterns, blocking frequencies, and annular mode timescales (Shaw & Perlwitz, 2010).

3.24.6 More about this POD

Sign of eddy heat fluxes in NH vs SH

In the Northern Hemisphere (NH), positive eddy heat fluxes represent poleward and upward wave fluxes. However, in the Southern Hemisphere (SH), the sign is flipped such that negative eddy heat fluxes represent the poleward and upward wave fluxes. This means that the SH polar cap eddy heat flux distributions will appear "flipped" compared to those for the NH. This also means that the extreme positive/negative heat flux events are in the opposite sense of those in the NH (i.e., extreme negative SH heat flux events are akin to extreme positive NH heat flux events).

Tip about horizontal resolution of data

Since this POD is primarily concerned with planetary scale waves, data with high horizontal resolution can be usefully downsampled without affecting results too much. This can speed up the MDTF data preprocessing and POD operation, while also decreasing the memory footprint.

3.24.7 References

Andrews, D. G., J. R. Holton, and C. B. Leovy, 1987: Middle Atmosphere Dynamics, Academic press, No. 40.

Charney, J. G., and P. G. Drazin, 1961: Propagation of planetary-scale disturbances from the lower into the upper atmosphere. Journal of Geophysical Research, 66(1), 83-109.

Dunn-Sigouin, E., and T. A. Shaw, 2015: Comparing and contrasting extreme stratospheric events, including their coupling to the tropospheric circulation. J. Geophys. Res. Atmos., 120: 1374– 1390. https://doi.org/10.1002/2014JD022116

England, M. R., T. A. Shaw, and L. M. Polvani, 2016: Troposphere-stratosphere dynamical coupling in the southern high latitudes and its linkage to the Amundsen Sea. Journal of Geophysical Research: Atmospheres, 121, 3776–3789, https://doi.org/10.1002/2015JD024254.

Hersbach, H. and coauthors, 2020: The ERA5 global reanalysis. Q J R Meteorol Soc., 146, 1999-2049, https://doi.org/10.1002/qj.3803

Randel, W. J., 1987: A Study of Planetary Waves in the Southern Winter Troposphere and Stratosphere. Part I: Wave Structure and Vertical Propagation. J. Atmos. Sci., 44, 917–935, https://doi.org/10.1175/1520-0469(1987)044<0917:ASOPWI>2.0.CO;2.

Shaw, T. A., J. Perlwitz, and N. Harnik, 2010: Downward Wave Coupling between the Stratosphere and Troposphere: The Importance of Meridional Wave Guiding and Comparison with Zonal-Mean Coupling. J. Climate, 23, 6365–6381, https://doi.org/10.1175/2010JCLI3804.1.

Shaw, T. A., and J. Perlwitz 2010: The Impact of Stratospheric Model Configuration on Planetary-Scale Waves in Northern Hemisphere Winter, J. Clim., 23(12), 3369-3389. https://doi.org/10.1175/2010JCLI3438.1

Shaw, T. A., J. Perlwitz, and O. Weiner, 2014: Troposphere-stratosphere coupling: Links to North Atlantic weather and climate, including their representation in CMIP5 models. J. Geophys. Res.: Atmospheres, 119, 5864–5880, https://doi.org/10.1002/2013JD021191.

3.25 TC MSE Variance Budget Analysis

Last Update: 2/22/2023

This POD computes the column-integrated moist static energy (MSE) and terms in the budget for its spatial variance for tropical cyclones (TCs). The budget terms are computed along the tracks of individual simulated TCs and them composited as a function of the TC intensity (maximum near-surface wind speed) of each snapshot. The results are compared to equivalent calculations from 5 reanalysis datasets.

3.25.1 Version & Contact info

- Version/revision information: version 1 (2/22/2023)
- PI: Allison Wing, Florida State University, awing@fsu.edu
- Developer/point of contact: Jarrett Starr, Florida State University, jstarr2@fsu.edu
- Other Contributors: Caitlin Dirkes, Suzana Camargo, Daehyun Kim

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt). .. Unless you've distributed your script elsewhere, you don't need to change this.

3.25.2 Functionality

In the current implementation of the POD, pre-calcluated tropical cyclone (TC) track data is also required as an input to the POD as obs data. The POD is currently written to accept track data as a formatted .txt file. The code extracts the TC center latitude and longitude, maximum near surface wind speed (vmax), and minimum sea level pressure at each time along the track of each storm. The code then extracts the necessary variables to compute the column-integrated moist static energy (MSE) and the longwave, shortwave, and surface flux feedbacks in the budget for the spatial variance of column-integrated MSE, in 10 x 10 degree boxes along the tracks of each TC. Then, the snapshots are trimmed to only account for times where the TC is intensifying (snapshots prior to each storms lifetime maximum intensity) and is equatorward of 30 degrees. The remaining TC snapshots are then binned by vmax in 3 m/s increment, and then composited over each bin. The model composites are then compared to 5 reanalysis datasets (ERA-5, ERA-Interim, MERRA-2, CFSR, JRA-55) which have already been processed through the above framework. The plots include composite TC-relative spatial maps of the feedback terms and MSE anomaly for select bins, azimuthal averages of the feedback terms for select bins, and box-averages of the feedback terms for all bins. A normalized version of the box-average plot is also included, in which the feedbacks at each grid point are normalized by the value of the box-average MSE variance for that snapshot. The normalization is performed prior to compositing. Finally, the box-averaged

feedbacks and normalized box-averaged feedbacks in select bins are plotted against the percent of storms intensifying from one bin to another.

When and how each of the scripts are utilized in the driver script (TC_MSE_Driver.py) is as follows:

1. TC_snapshot_MSE_calc.py is called first to extract the data and compute the MSE variance budget along the tracks of all the TCs. The resulting data is saved into a file for each year.

2. Binning_and_compositing.py is called which takes all of the files that were created in step 1, concatenates them, and then bins as well as composites each of the snapshots and its variables by vmax. The budget variables are also box-averaged and normalized in this step as well.

3. Plotting.py is called which imports all of the plotting functions that are in the Plotting_Functions.py script and generates and saves the plots that are desired. The user may comment out any of the plotting functions that are called in the Plotting.py script they do not want.

3.25.3 Required programming language and libraries

Written using conda version 4.10.1 and python version 3.10

matplotlib, numpy, pandas, xarray, scipy, sys

3.25.4 Required model output variables

The following 3-D (time-lat-lon) and 4-D (time-plev-lat-lon) variables are required:

3-D Variables:

Surface Downwelling Longwave Flux in Air (rlds), Units: W m-2, Frequency: 6-hourly Surface Downwelling Shortwave Flux in Air (rsds), Units: W m-2, Frequency: 6-hourly Surface Upwelling Longwave Flux in Air (rlus), Units: W m-2, Frequency: 6-hourly Surface Upwelling Shortwave Flux in Air (rsus), Units: W m-2, Frequency: 6-hourly Surface Upward Latent Heat Flux (hfls), Units: W m-2, Frequency: 6-hourly Surface Upward Sensible Heat Flux (hfss), Units: W m-2, Frequency: 6-hourly Top of Atmosphere Outgoing Longwave Flux (rlut), Units: W m-2, Frequency: 6-hourly Top of Atmosphere Outgoing Shortwave Flux (rsut), Units: W m-2, Frequency: 6-hourly Top of Atmosphere Incoming Shortwave Flux (rsdt), Units: W m-2, Frequency: 6-hourly

4-D Variables:

Air Temperature (ta), Units: K, Frequency: 6-hourly instantaneous Geopotential Height (zg), Units: m, Frequency: 6-hourly instantaneous Specific Humidity (hus), Units: g g-1, Frequency: 6-hourly instantaneous

In order to have sufficient samples, we recommend enough years of data to have at least 100 TCs, which is typically 2-5 years of simulation time depending on the resolution of the model.

In the current implementation of the POD, pre-calcluated TC track data is also required as an input to the POD as obs data. If future versions incorporate TC tracking directly, additional model output variables will be required.

3.25.5 References

1. Wing, A. A., Camargo, S. J., Sobel, A. H., Kim, D., Moon, Y., Murakami, H., Reed, K. A., Vecchi, G. A., Wehner, M. F., Zarzycki, C., & Zhao, M. (2019). Moist Static Energy Budget Analysis of Tropical Cyclone Intensification in High-Resolution Climate Models, Journal of Climate, 32(18), 6071-6095, https://doi.org/10.1175/JCLI-D-18-0599.1.

2. Dirkes, C. A., Wing, A. A., Camargo, S. J., Kim, D. (2022). Process-oriented diagnosis of tropical cyclones in reanalyses using a moist static energy variance budget, Journal of Climate (In review).

3.25.6 More about this diagnostic

The MSE variance budget has been shown to capture the role of important physical processes in the development of TCs (Wing et al. 2016; Wing et al. 2019; Wing 2022). A budget for the spatial variance of column-integated MSE is given by:

$$\frac{1}{2}\frac{\partial \hat{h}'^2}{\partial t} = \hat{h}'F_k' + \hat{h}'N_L' + \hat{h}'N_S' - \hat{h}'(\widehat{\vec{u}\cdot\nabla h})'.$$

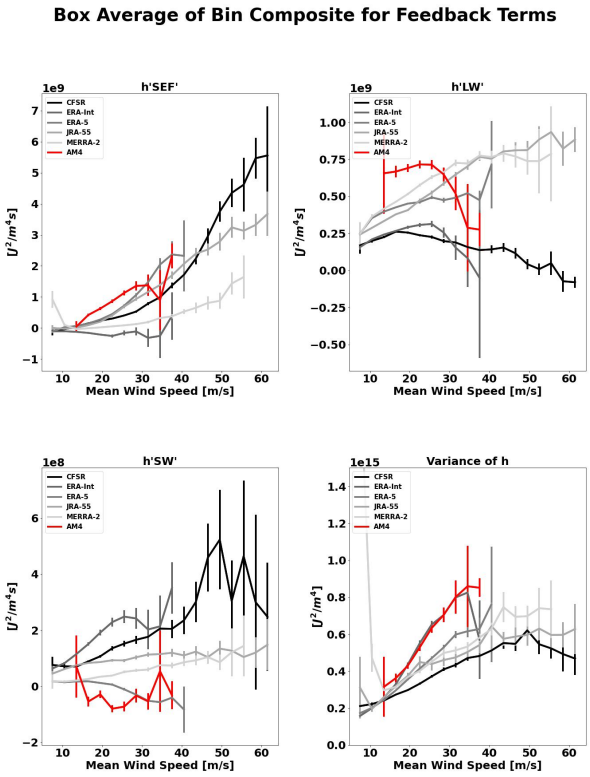
The hat notation indicates a mass-weighted column integral. There are three diabatic feedback terms (on the RHS) which from left to right are the surface flux, longwave, and shortwave feedback respectively. The far right feedback is the advective term which we do not calculate in this POD as we focus on the three diabatic terms only. Each of the feedback terms are considered sources and sinks of MSE variance. The prime notation indicates the anomaly from the average value of the given variable over a box centered on the TC. For example, for each snapshot along the track of each TC, the column-integrated MSE ($\hat{h}$) is calculated as the anomaly of the column-integrated MSE at each grid point within a 10 x 10 degree box centered on the TC from its cosine-latitude weighted mean over that box. Anomalies are also computed for each of the diabatic sources and sinks of column-integrated MSE: surface enthalpy flux (F_k'), column longwave radiative flux convergence (N_L'), and the column shortwave radiative flux convergence (N_S').

Sources of MSE variance are seen when a feedback term is positive, which occurs when an individual grid point has anomalies of column-interated MSE and anomalies of its source/sink of the same sign. That is, if there is an anomalous source of column-integrated MSE at the same place where the column-integrated MSE is already anomalously large, this will amplify the MSE anomaly and act to increase the MSE variance. Since the spatial variability of MSE in the tropical atmosphere (even in TCs) is dominated by moisture variability, an increase in MSE spatial variance reflects that the moist regions get moister and the dry regions get drier. Conversely, sinks of MSE variance are termed negative feedbacks, where dry regions are being moistened and moist regions are being dried.

As an example, in the areas closest to the TC center, column-integrated MSE will tend to be above the box average value giving a positive value of $\hat{h}'$. In the event that a model or reanalysis has the ability to detect an eye-like feature for a TC, the winds will be more calm in the center giving rise to a smaller surface enthalpy flux and a negative value of F_k'. Therefore, in this scenario we would observe the surface flux feedback term being a sink of MSE variance in that location. This can be observed in the spatial composite plots of the finer horizontal resolution reanalyses that pick up on the eye-like features in TCs. Since MSE variance increases with TC intensity, sources of MSE variance play can be interpreted as contributing to TC development (Wing et al. 2019; Dirkes et al. 2022).

In order to compare across TC snapshots of similar intensity, we bin the MSE variance and its diabatic feedbacks based on the maximum near surface wind speed of the TC (vmax). Wing et al. (2019) noted that if we composite based on time leading up to LMI, the strength of the feedbacks will vary in a given snapshot as the LMI and preceding vmax for each storm can be vastly different between any given storm, and across models or reanalyses. Therefore, by binning in 3 m/s bins we can compare the feedback representation at a given intensity across different mdels or between models and reanalyses.

The plots provided as part of this POD follow those from Wing et al. (2019) and Dirkes et al. (2022). The spatial composite panel plotting allows one to see a given model's spatial representation of the three diabatic feedback terms in comparison with 5 modern reanalyses, for one intensity bin. The azimuthal mean plots reduce the dimensionality and reveal how the feedbacks vary with radial distance from the TC center in each intensity bin. What can be noted is that as the distance from the TC center increases, all the feedbacks tend to approach zero which results from being too far from the influence of the TC. Further reducing the dimensionality, the box-averaged feedback intensity bin composites show how the average of the feedback over the entire TC-centered box depend on TC intensity, and for a given intensity, the model can becompared to the reanalyses. It can be noted that as a storm increases in intensity, the areas nearest the TC become warmer and moister, and thus the variance of column-integrated MSE increases. The three diabatic feedbacks also tend to increase with intensity. Error bars indicate the 5-95% confidence interval for the box-average feedback. The values are only plotted if there are at least two samples in that bin, though in practice more than 50 samples must be present for the error bars to be small. Thus, the first bin that is plotted is the 6 to 9 m/s bin but the models and reanalyses are best compared between intensities of 10 and 30 m/s. An example of this plot is shown below where the CMIP6 GFDL CM4 AMIP simulation (doi:10.22033/ESGF/CMIP6.8494) is compared to the reanalyses.



The normalized version of these box-averaged feedbacks are also calculated which is done by dividing the feedback at each grid point by the box-averaged column-integrated MSE variance, prior to doing the intensity bin composites. This is done to remove the effect of column-integrated MSE anomalies increasing as a storm is intensifying, as well as account for different baseline representations of MSE variability across different models/reanalyses. The normalized feedbacks can be interprted as a growth rate of MSE variance per day.

The final plot relates the percentage of storms that intensify from one intensity bin to the next to the value of the diabatic feedbacks at the starting intensity. This can be used to help attribute intermodel variability in TC intensification to process representation as quantified by the MSE variance budget feedbacks. For example, Wing et al. 2019 showed that models that had a larger surface flux feedback at a given intensity produced more intense TCs on average. Across reanalyses, Dirkes et al. 2022 showed that reanalyses that had a larger longwave feedback at a given intensity produced a greater fraction of TCs that intensified further.

Additional References

1. Wing, A.A., S.J. Camargo, and A.H. Sobel (2016), Role of radiative-convective feedbacks in spontaneous tropical cyclogenesis in idealized numerical simulations, J. Atmos. Sci., 73, 2633-2642, doi:10.1175/JAS-D-15-0380.1.
2. Wing, A.A. (2022): Acceleration of tropical cyclone development by cloud-radiative feedbacks, J. Atmos. Sci., 79, 2285–2305, doi:10.1175/JAS-D-21-0227.1.

3.26 TC Rain Rate Azimuthal Average Documentation

Last update: 5/27/2022

This POD calculates and plots azimuthal averages for tropical cyclone (TC) rain rates from TC track data and model output precipitation.

3.26.1 Version & Contact info

- Version/revision information: version 1 (5/06/2020)
- PI Daehyun Kim, University of Washington, daehyun@uw.edu
- Developer/point of contact Nelly Emlaw, University of Washington, gnemlaw@uw.edu

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt). Unless you've distributed your script elsewhere, you don't need to change this.

3.26.2 Functionality

In the current draft code, the data is loaded in first. The netcdf file for the total precipitation is loaded in through xarray. The TC track data is read in line by line from a txt file.

Then the azimuthal average is calulated for a set of discrete radii by measuring the distance between a data point and the center of the storm for each snapshot.

A plot showing TC rain rate as a function of radius is made for select snapshopts that reach 35-45 knot max wind speeds.

3.26.3 Required programming language and libraries

matplotlib, numpy, netcdf, xarray, scipy

3.26.4 Required model output variables

Total Precipitation

TC Track Data (required: storm center latitude and longitufe, time for each snapshot, optional: max windspeed, minimum central surface pressure, sst, etc)

3.26.5 References

1. Kim, D., Y. Moon, S. Camargo, A. Sobel, A. Wing, H. Murakami, G. Vecchi, M. Zhao, and E. Page, 2018: Process-oriented diagnosis of tropical cyclones in high-resolution climate models. J. Climate, 31, 1685–1702, https://doi.org/10.1175/JCLI-D-17-0269.1.

2. Moon, Y., D. Kim, S. Camargo, A. Wing, A. Sobel, H. Murakami, K. Reed, G. Vecchi, M. Wehner, C. Zarzycki, and M. Zhao, 2020: Azimuthally averaged wind and thermodynamic structures of tropical cyclones in global climate models and their sensitivity to horizontal resolution. J. Climate, 33, 1575–1595, https://doi.org/10.1175/JCLI-D-19-0172.1

3. Moon, Y., D. Kim, A. A. Wing, S. J. Camargo, M. Zhao, L. R. Leung, M. J. Roberts, D.-H. Cha, and J. Moon: An evaluation of global climate model-simulated tropical cyclone rainfall structures in the HighResMIP against the satellite observations, J. Climate, Accepted.

3.26.6 More about this diagnostic

This POD calculates the azimuthally averaged precipitation rate of tropical cyclones (TCs) as a function of distance from the center of a TC from hourly (1,3,6, or 12 hr) model precipitation output and model TC track output.

In its current version the POD requires TC track input and cannot track TCs from model output alone. The track data must provide the latitude and longitude of the storm center and the time of track snapshot and optionally a characteristic of the storm at that snapshot to use as a threshold to use the storm in the plotted average (here max wind is used and the default threshold is set to 35-45 knots to capture weak storms which meet the requirements to be considered a tropical cyclone - this threshold can be changed in the setting.jsonc file in the POD directory).

The POD takes track data separated by basin. The basin longitude regions are determined by the ECMWF TC tracking standards and are as follows: Atlantic Ocean (atl) : 100 - 350 W Eastern Central Pacific (enp) : 0 - 100 W Western North Pacific (wnp) : 100 - 180 E Indian Ocean (nin : 40 - 100 E South Indian Ocean (sin) : 30 - 90 E Australian Ocean (aus) : 90 - 160 East South Pacific Ocean (spc) : 160 E - 120 W The latitude range for all basins is 0 - 30 N or S depending on the hemispehre.

The output of this POD will be a plot of the azimuthally averaged rain rate (vertical axis) as a fuction of distance from the center of the storm (horizontal axis). The result should always be the highest rain rates at or just off center of the storm. Rain rate distributions with greatest inner-core rainfall tend to simulate stronger TCs more often. The distribution will vary greatly depending on model characteristics and threshold to determine snapshops used in average. For example was speculated in Moon et. al, 2020 that low resolution models require more energy to sustain themselves through induced precipitation.

3.27 Surface Temperature Extremes and Distribution Shape Package

Last update: 7/7/20

The surface temperature extremes and distribution shape package computes statistics that relate to the shape of the two-meter temperature distribution and its influence on extreme temperature exceedances. These metrics evaluate model fidelity in capturing moments of the temperature distribution and distribution tail properties, as well as the large-scale meteorological patterns associated with extreme temperature exceedance days. Statistics include temperature mean, standard deviation, and skewness, along with PDFs of the underlying temperature distribution and the Gaussian fit to the distribution core of these values to identify non-Gaussianity at a location. Frequency of exceedances of a fixed extreme temperature threshold after a uniform warm shift across the distribution, the simplest prototype for future warming, provides a measure of Gaussianity as well as insight into the complexity of future changes in temperature extremes.

3.27.1 Version & Contact Information

- Version 1 07-Jul-2020 Arielle J. Catalano (Portland State University)
- PI: J. David Neelin (UCLA; neelin@atmos.ucla.edu)
- Science lead: Paul C. Loikith (PSU; ploikith@pdx.edu)
- Current developer: Arielle J. Catalano (PSU; a.j.catalano@pdx.edu)

Open Source Copyright Agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.27.2 Functionality

This package consists of the following functionalities:

1. Moments of Surface Temperature Probability Distribution (TempExtDistShape_Moments.py)
2. Shifted Underlying-to-Gaussian Distribution Tail Exceedances Ratio (TempExtDistShape_ShiftRatio.py)
3. Frequency Distributions at Non-Gaussian Tail Locations (TempExtDistShape_FreqDist.py)
4. Composite Circulation at Non-Gaussian Tail Locations (TempExtDistShape_CircComps.py)

As a module of the MDTF code package, all scripts of this package can be found under `../diagnostics/temp_extremes_distshape/` and predigested observational data under `../inputdata/obs_data/temp_extremes_distshape/`.

3.27.3 Required Programming Language and Libraries

This package is written in Python 3, and requires the following Python packages: os, json, numpy, scipy, matplotlib, mpl_toolkits, h5py, netCDF4, netcdftime, cftime, cartopy.

These Python packages are already included in the standard Anaconda installation.

3.27.4 Required Model Output Variables

The following 3-D (lat-lon-time) model fields at a daily resolution are required:

1. Two-meter temperature (units: K or °C)
2. Sea level pressure (units: hPa or Pa)
3. Geopotential height (units: m)

Recommended timeseries length to facilitate statistical robustness for main features: Preferred is 30 years matching the observational analysis; Minimum is 15 years.

Observational Data Summary

Gridded two-meter temperature, sea level pressure (SLP), and 500 hPa geopotential height (Z500) data are from the Modern Era Retrospective-Analysis for Research and Applications reanalysis version 2 (MERRA-2; Gelaro et al. 2017, https://doi.org/10.1175/JCLI-D-16-0758.1). Daily means constructed from hourly data at the native horizontal resolution 0.5° × 0.625° are provided as observational data. Analyses of MERRA-2 use the 30-year period 1980–2009 following *Loikith and Neelin 2019* (page 119).

This POD can be run directly on MERRA-2 data when set as the casename and input model in the default_tests.jsonc file with netcdf files supplied in the `../inputdata/model/` folder (within its own named folder "temp_extremes_distshape" per general usage instructions). To rerun MERRA-2 data with different default settings (see below) change variables under "pod_env_vars" in settings.jsonc.

Default Settings Options

Location: Frequency distributions and composite circulation maps can be output for user-specified locations. By default, the six selected locations for cold-tail frequency distributions are Yellowknife, Pendleton, Rennes St.-Jacques, Berlin, Adelaide, and North Platte (see *Loikith and Neelin 2019* (page 119)), and the selected location for composite circulation maps is Yellowknife. Changes to locations for the frequency distributions can be made in the User Specified Section of the "...usp.py" file (TempExtDistShape_FreqDist_usp.py), and the selected location for composite maps can be altered in settings.jsonc ("city"). Note: plotting parameters for composite circulation maps may need to be altered to encompass the range of data at an alternate location, and are located in the "..usp.py" file.

Season: By default, the POD outputs cold-tail figures for the season spanning the months of December, January, and February (DJF). To output figures for the season spanning June, July, and August (JJA), change the "monthstr" and "monthsub" variables in settings.jsonc. The "monthstr" value is the string of letters corresponding to the names ('JJA' or 'DJF'), whereas "monthsub" is the array of integers ([6,7,8] or [1, 2, 12]). Note: this changes the output figures for three of the functionalities only. The figure for cold-tail frequency distributions at non-Gaussian locations includes examples in both seasons and thus remains unchanged. Changes to locations and seasons in this functionality must be made within the User Specified Section of the source code (TempExtDistShape_FreqDist_usp.py).

Tail: By default, the POD outputs figures for the cold-side tail of the surface temperature distribution, using the 5th percentile as the threshold from which to measure exceedances. To output figures for the warm-side tail (95th percentile), change the "ptile" percentile value in settings.jsonc. The only functionalities affected by this change are the shifted underlying-to-Gaussian distribution tail exceedances ratio and the composite circulation figure. This is because 1) the moments of the temperature distribution are for the full distribution and not just the tail, and 2) non-Gaussian tail locations selected for the frequency distribution function are specifically chosen to represent shorter-than-Gaussian and longer-than-Gaussian cold tails following *Loikith and Neelin 2019* (page 119).

3.27.5 References

Ruff, T. W., and J. D. Neelin, 2012: Long tails in regional surface temperature probability distributions with implications for extremes under global warming. *Geophys. Res. Lett.*, **39**, L04704, https://doi.org/10.1029/2011GL050610.

Loikith, P. C., and J. D. Neelin, 2015: Short-tailed temperature distributions over North America and implications for future changes in extremes. *Geophys. Res. Lett.*, **42**, https://doi.org/10.1002/2015GL065602.

Loikith, P. C., J. D. Neelin, J. Meyerson, and J. S. Hunter, 2018: Short warm-side temperature distribution tails drive hotspots of warm temperature extreme increases under near-future warming. *J. Climate*, **31**, 9469–9487, https://doi.org/10.1175/JCLID-17-0878.1.

Loikith, P. C., and J. D. Neelin, 2019: Non-Gaussian cold-side temperature distribution tails and associated synoptic meteorology. *J. Climate*, **32**, 8399–8414, https://doi.org/10.1175/JCLI-D-19-0344.1.

Catalano, A. J., P. C. Loikith, and J. D. Neelin, 2020: Evaluating CMIP6 model fidelity at simulating non-Gaussian temperature distribution tails. *Environ. Res. Lett.*, **15**, 074026, https://doi.org/10.1088/1748-9326/ab8cd0.

3.27.6 More About This Diagnostic

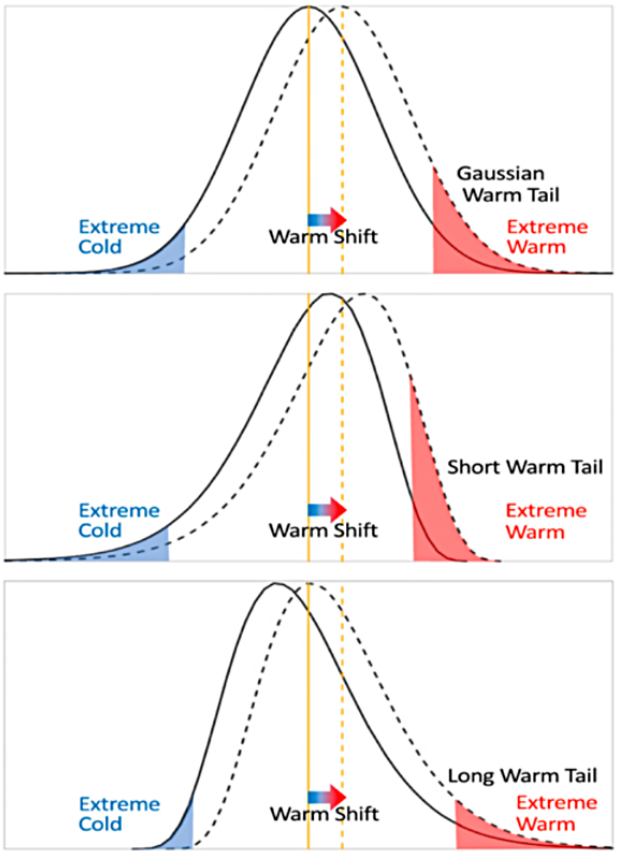


Fig. 9: A schematic illustrating the effect of non-Gaussian tails on changes in the frequency of exceeding a fixed extreme temperature threshold under a uniform warm shift. Solid curves are the temperature frequency distributions for the pre-shifted climate while dotted curves are the temperature frequency distributions after the warm shift. Vertical yellow lines are pre-shift (solid) and post-shift (dashed) means. The red shaded area represents temperatures above the 95th percentile of the preshifted distribution while the blue shaded area represents the 5th percentile. Illustrations are for (top) a Gaussian, (middle) a distribution with a shorter-than-Gaussian warm tail and a longer-than-Gaussian cold tail, and (bottom) a distribution with a longer-than-Gaussian warm tail and a shorter-than-Gaussian cold tail.

The current version of the surface temperature extremes and distribution shape package produces four sets of figures for both pre-digested observations and model output, including (1) moments of the distribution (i.e. mean, variance, skewness), (2) shift ratio, (3) frequency distributions, and (4) composite large-scale meteorological patterns. The shift ratio is computed by shifting the underlying daily temperature distribution uniformly by a fixed amount, tabulating the frequency of days that exceed a fixed pre-shifted extreme temperature threshold, and dividing that by the expected number of threshold exceedances from shifting a Gaussian by the same amount. For example, if the underlying distribution has a short warm tail, exceedances will be greater under a uniform warm shift than for a Gaussian distribution. Thus, the shift ratio identifies regions with a shorter- or longer-than-Gaussian tail. Locations exhibiting a non-Gaussian tail can be selected for further analysis of frequency distributions and composite circulation maps. In the following, we will show an example set of the figures for a simulation of the GFDL-CM3 model (included in Phase 5 of the IPCC Coupled Model Intercomparison Project) that are produced by the package.

1) Moments of the Surface Temperature Probability Distribution

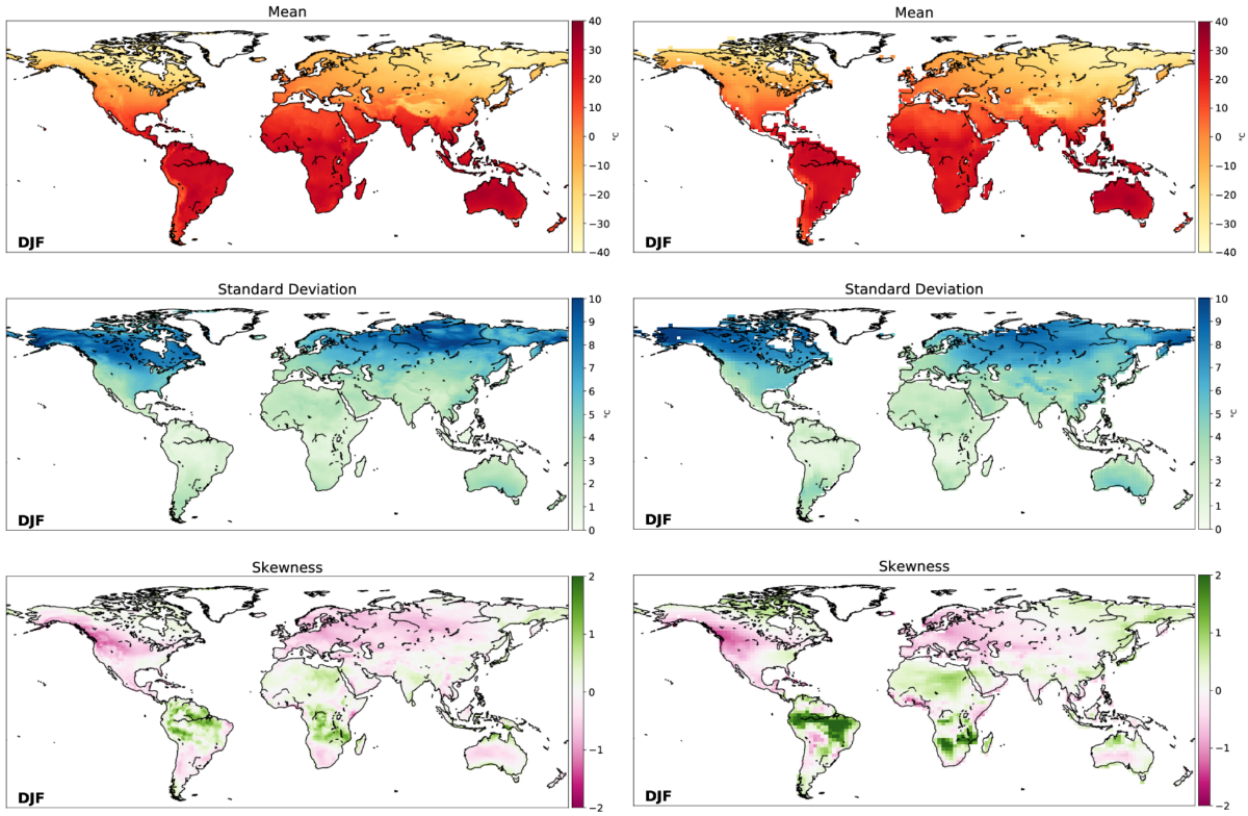


Fig. 10: Moments for the DJF daily two-meter temperature distributions (°C) including the (top) mean, (middle) standard deviation, and (bottom) skewness. The left column are results from MERRA-2, and the right column are results from GFDL-CM3.

The three moments of the temperature distribution characterize its shape by measuring the central tendency, spread, and symmetry. During DJF, mean temperatures are coolest in the higher latitudes and altitudes of the Northern Hemisphere and warmest in the Southern Hemisphere. The standard deviation is largest in Northern North America and Eurasia owing to synoptic variability. Skewness measures the asymmetry of the full distribution of two-meter temperatures at each location, with positive values generally indicating a longer cold tail and negative indicating a shorter cold tail. Positively skewed regions include northern Africa, eastern Siberia, and the tropical regions of South America and Africa. Negatively skewed regions include South Africa, central Australia, Eurasia, and western North America. There is broad spatial agreement between GFDL-CM3 and MERRA-2 in the mean and standard deviation of two-meter temperature distributions over DJF, although the model does not capture some fine-scale features of standard deviation over Siberia and Yukon, which may be a result of the lower spatial resolution. GFDL-CM3 captures regions of negative skewness over western North America and Europe, and positive skewness in tropical South America and Africa, though the magnitude is higher in GFDL-CM3. Also, the model does not adequately simulate skewness direction over some regions including Northern Russia and Southern India, which may affect how future changes in extreme temperature exceedances are manifested.

2) Shift Ratio

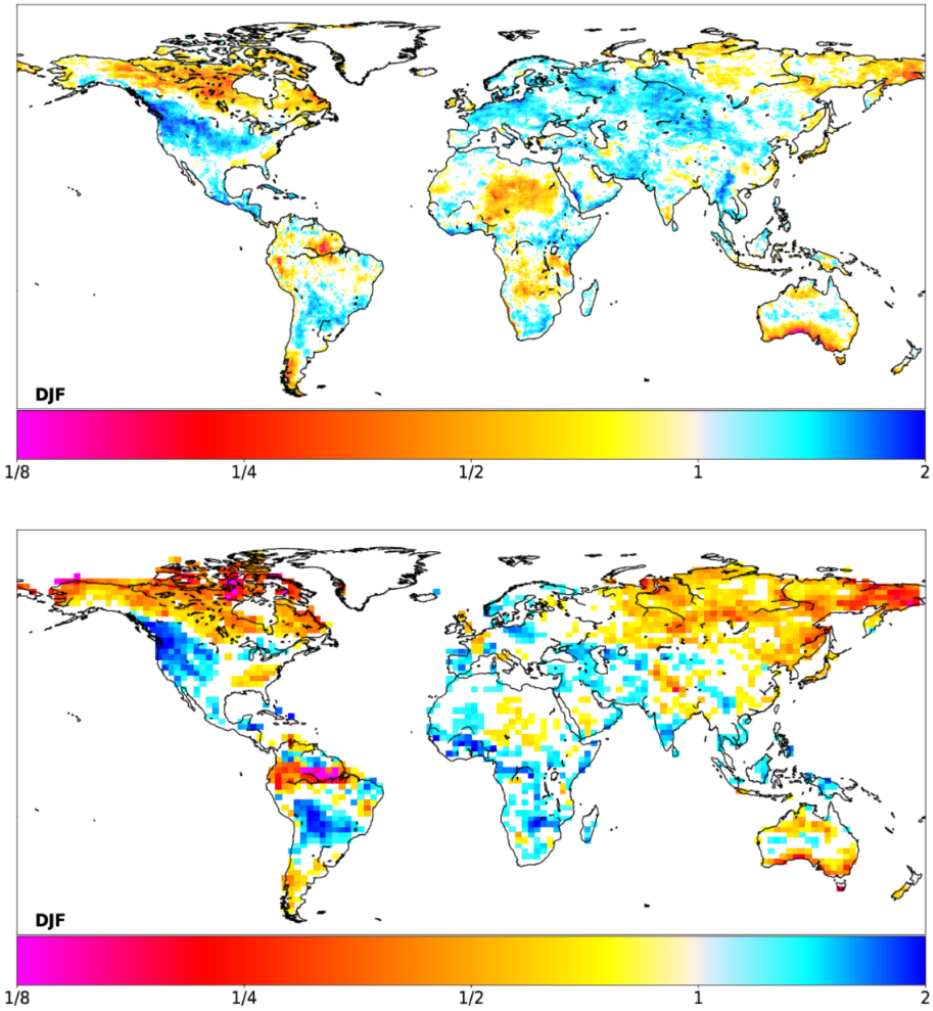


Fig. 11: Shift ratio for cold tails during DJF, using the 5th-percentile as the fixed threshold to measure exceedances, in (*top*) MERRA-2 and (*bottom*) GFDL-CM3. Values greater than one indicate longer-than-Gaussian cold tails and values less than one indicate shorter-than-Gaussian cold tails. Only grid cells where the cold tail deviates from Gaussian with statistical significance at the 5% confidence level are shaded.

While skewness characterizes the asymmetry of the full distribution, the shift ratio provides an understanding of non-Gaussianity specific to the tail. The shift ratio identifies regions of non-Gaussianity and regions statistically indistinguishable from Gaussian based on a ratio of distribution tail exceedances following the application of a uniform shift (0.5). This procedure acts as both a pragmatic way of demonstrating the effects of non-Gaussian tails on extremes exceedances and also provides a statistically robust measure of tail departure from Gaussian. Coherent regions with longer-than-Gaussian cold tails during DJF, such as northwestern United States, will experience a smaller decrease in 5th-percentile exceedances under future warming than regions with shorter-than-Gaussian cold tails (e.g., southern Australia). GFDL-CM3 captures spatially coherent short-tailed regions such as southern Australia, Canada, and northeastern Siberia, as well as long-tailed regions such as northwestern United States and central South America, although ratio amplitude is higher in GFDL-CM3. GFDL-CM3 tail shape differs from MERRA-2 over Europe and western Russia, as values are greater than one in MERRA-2 but below or insignificant in the model. This region also differs between datasets in skewness direction (see Moments of the Distribution section above), which is a related measure of asymmetry. These differences suggest that the model may simulate a smaller and potentially slower decline in DJF extreme cold temperatures over this region under future warming.

3) Frequency Distributions

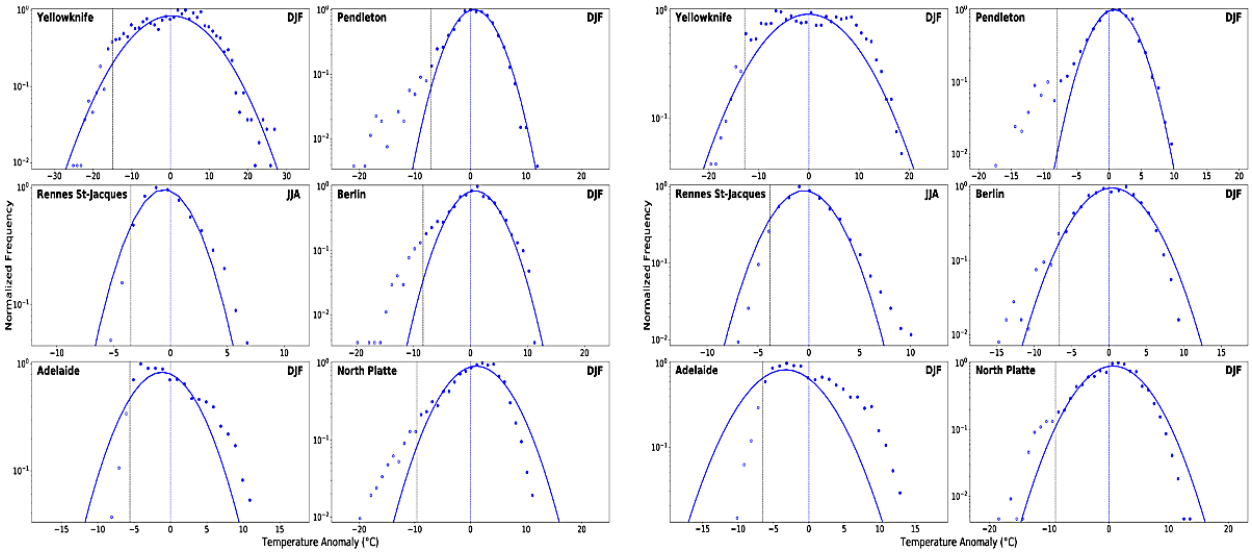
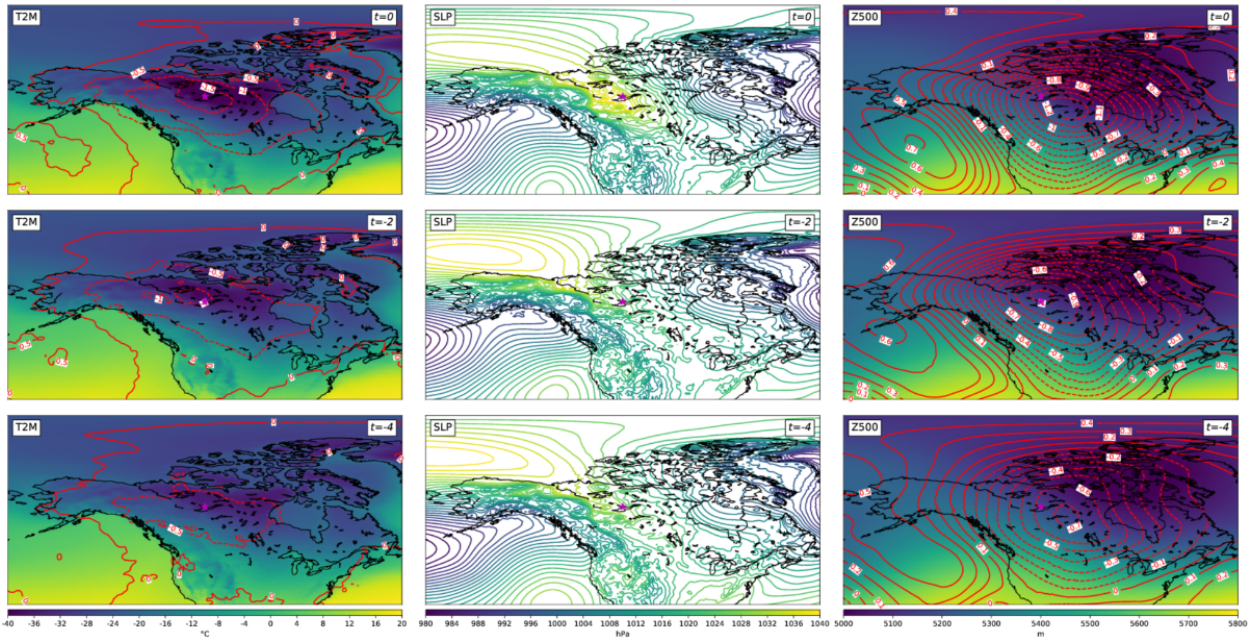


Fig. 12: Frequency distributions of two-meter temperature at grid cells nearest to select non-Gaussian locations in (left) MERRA-2 and (right) GFDL-CM3. Circles represent temperatures binned in 1°C intervals, with open circles identifying the cold tail of the distribution below the 5th percentile (black dashed line). The dashed blue line is the distribution mean, and the solid blue line is the Gaussian curve fit to the core of the distribution (values greater than 0.3 of the maximum). Frequency is provided on a log scale to highlight the tails. Short-tailed locations during respective seasons include: Yellowknife, Canada; Rennes St-Jacques, France; and Adelaide, Australia. Long-tailed locations include: Pendleton, United States; Berlin, Germany; and North Platte, United States. Note, the Gaussian at Yellowknife is fit to the entire distribution rather than the core as the Gaussian core fit was too wide, the result of which masks short-tail behavior (see Loikith et al. 2019 for more details).

Normalized frequency distributions of two-meter temperature anomalies provide a closer examination of distribution shape at selected locations exhibiting non-Gaussian behavior in the tail in the MERRA-2 distributions. A Gaussian curve is fit to the core of the underlying temperature distribution by polynomial regression, with the core being defined as values exceeding a threshold of 0.3 of the distribution maximum (Ruff and Neelin 2012). The fitted curve highlights the departures from Gaussianity in tails at select locations for the season specified. For example, the temperature distribution at Rennes St-Jacques, France exhibits a shorter-than-Gaussian cold-side tail during JJA, with bin counts (open circles) less than those estimated from the curve. Conversely, Pendleton, United States exhibits a longer-than-Gaussian cold-side tail during DJF, with binned anomalies in the cold tail nearly 10 degrees cooler than those estimated from the Gaussian fit. Overall, GFDL-CM3 simulates appropriate cold-tail non-Gaussianity, but there are some subtle differences in distributions. The Gaussian curve at Adelaide, Australia is narrower in MERRA-2 than in GFDL-CM3, indicating a flatter temperature distribution core simulated by the model (to which the curve is fit). Also, the simulated distribution at Yellowknife, Canada underestimates the extreme temperatures experienced, as values in MERRA-2 are around ± 30°F but closer to ± 20°F in GFDL-CM3. These differences influence the standard deviation of the distribution at each location, which is integral to the shift applied for computing the ratio of tail exceedances under this simplest prototype for future warming.

4) Composite Circulation



After identifying areas exhibiting shorter-than-Gaussian or longer-than-Gaussian tails, composites of large-scale meteorological patterns lagged from tail exceedance days can be mapped to help diagnose the physical processes underlying the tail behavior. Large-scale meteorological patterns include surface temperature (T2M) and anomalies, SLP, and Z500 and anomalies for zero, two, and four days prior to tail exceedance days. The inclusion of these composites in the package is also process-oriented in that it enables the user to hypothesize possible reasons their model may or may not capture particular features. The user can specify the location at which to composite (see "Options…" below).

Here, the location example Yellowknife is used, where the DJF temperature distribution exhibits a shorter-than-Gaussian cold-side tail. Four days prior to an average cold-tail exceedance day, temperatures in the vicinity of Yellowknife are approximately -30°C, with anomalies around 0.5 below the climatological average. An area of high pressure is present north of Alaska and extending southeastward towards Yellowknife, and Z500 anomalies indicate a broad trough is present centered east of Yellowknife. By t=0, temperature anomalies at the location exceed 1.5, SLP values have increased substantially, and the Z500 trough to the southeast has deepened. The progression of these patterns on days prior to the cold-tail exceedance day indicates that the coldest temperatures within this area of short tails are not primarily the product of transport of air from much colder places, but rather conditions that promote the development of anomalously cold air in place (Loikith and Neelin 2019). As Yellowknife is situated near the coldest temperatures in the hemisphere, there is a limit on advection of highly anomalous cold air, which explains the short tails in this region.

GFDL-CM3 generally captures the features of the circulation and surface temperatures associated with DJF cold days at Yellowknife at each lag time, which supports the notion that the model is producing a shorter-than-Gaussian tail for plausible physical reasons. Notable differences include lower SLP values in the model, particularly at t=0 when the high pressure is not centered on the location but rather located north of Alaska, as well as the positioning of the Z500 trough, which is centered over Yellowknife in the model. These discrepancies in general circulation features could lead to potential biases in distribution shape characteristics, which influence how temperature extremes are manifested under climate change.

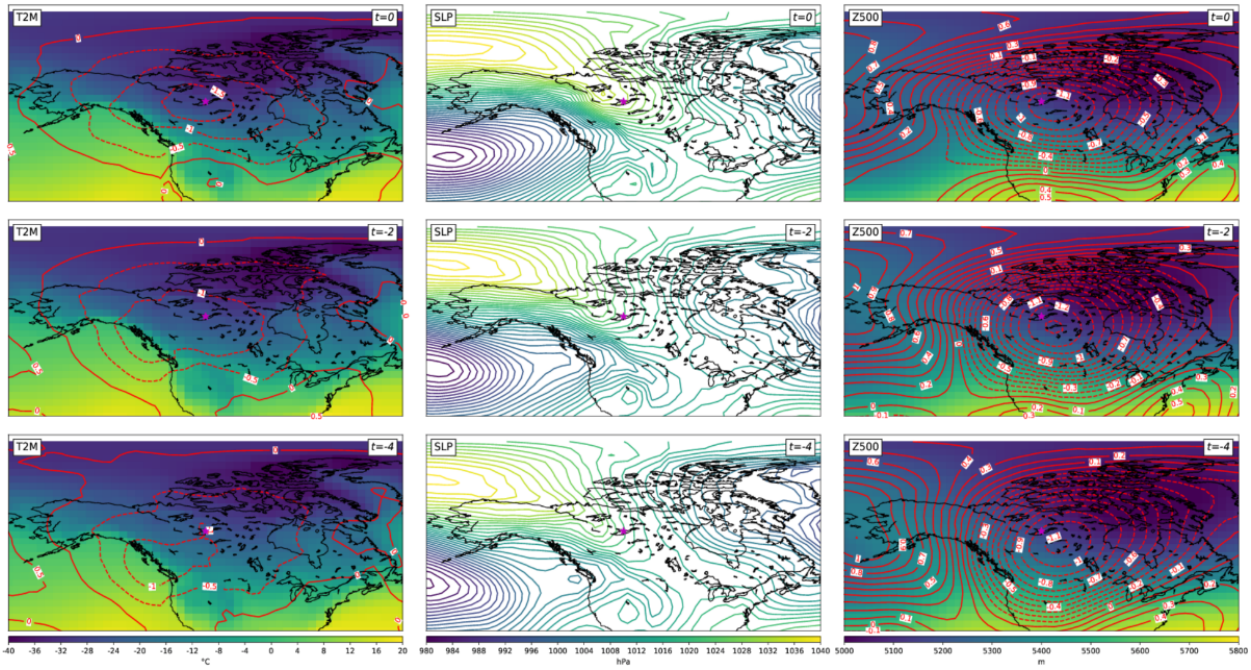


Fig. 13: Composite mean of large-scale meteorological patterns in (top) MERRA-2 and (bottom) GFDL-CM3 at Yellowknife (magenta star) for days below the 5th percentile of the temperature distribution (t=0), two days prior to these exceedances (t=-2), and four days prior (t=-4) during DJF. Left column shading represents two-meter temperature (°C), middle column contours represent sea level pressure (hPa), and right column shading represents 500-hPa geopotential heights (m). Red contours in left and right columns are temperature and geopotential height standardized anomalies, respectively.

3.28 Top-Heaviness Metric Diagnostic Documentation

Last update: 5/30/2021

The vertical profiles of diabatic heating have important implications for large-scale dynamics, especially for the coupling between the large-scale atmospheric circulation and precipitation processes. We adopt an objective approach to examine the top-heaviness of vertical motion (Back et al. 2017), which is closely related to the heating profiles and a commonly available model output variable. The diagnostic metric can also be used to evaluate the diabatic heating profile.

3.28.1 Version & Contact info

- Version/revision information: version 1.0 (6/28/2021)
- Developer/point of contact (Jiacheng Ye, jye18@illinois.edu, DAS UIUC; Zhuo Wang, zhuowang@illinois.edu, DAS UIUC)

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt).

3.28.2 Functionality

The current package consists of following functionalities:

(1) Calculation of the fractional variance of vertical velocity at each grid point explained by two base functions, Q1 (~idealized deep convection profile) and Q2 (~idealized deep stratiform profile)

(2) Calculation of the top-heaviness ratio (O2/O1)

As a module of the MDTF code package, all scripts of this package can be found under `mdtf/MDTF_$ver/diagnostics/top_heaviness_ratio`

3.28.3 Required programming language and libraries

Python3 packages: “netCDF4”, “xarray”, “numpy”, “scipy”, “matplotlib”, “cartopy”

3.28.4 Required model output variables

1) wap (plev x lat x lon) : Vertical Velocity [Pa/s], which can be either the monthly mean for a certain year or the long-term monthly/season mean.

3.28.5 References

Back, L. E., Hansen, Z., & Handlos, Z. (2017). Estimating vertical motion profile top-heaviness: Reanalysis compared to satellite-based observations and stratiform rain fraction. Journal of the Atmospheric Sciences, 74(3), 855-864. https://doi.org/10.1175/JAS-D-16-0062.1

Jiacheng and Zhuo's paper is in preparation.

3.28.6 More about this diagnostic

Q1 and Q2 (Figure 1a) are two prescribed base functions. Following Back et al. (2017), Q1 as a half sine function, and Q2 as a full sine function, which represent the idealized deep convection profile and the idealized stratiform profile, respectively. The vertical velocity can be approximated by Q1 and Q2:

'(x,y,p) = O1(x,y) * Q1(p) + O2(x,y)*Q2(p)

where O1 and O2 are the coefficients of Q1 and Q2, respectively. Back et al. (2017) showed that Q1 and Q2 resemble the first two EOF modes of vertical velocity profile variability. Then O1 and O2 can be approximately regarded as the corresponding principal component time series.

For O1>0, ' transitions from a bottom-heavy profile to a top-heavy profile when the ratio of r=O2/O1 increases from -1 to 1 (Figure 1b). The ratio r is thus defined as the top-heaviness ratio.

To assess how well ' approximates , the fractional variance is calculated over each grid point. The fractional variance is defined as the square of the pearson correlation between ' and . As shown in Figure 2, ' explains more than 80% of the vertical variance over most tropical/subtropical oceanic grid points.

The top-heaviness ratio (r) is presented in Figure 3. The Western Pacific is dominated by more top-heavy vertical profiles while the Eastern Pacific and Atlantic are characterized by more bottom-heavy profiles, exhibiting a great contrast.

3.29 Tropical Pacific Sea Level Diagnostic Documentation

Last update: 11/16/2020

Sea level rise is closely related to climate variability and change. It has important socio-economic impacts on many coastal cities and island nations. The largest sea level variability occurs in the tropical Pacific, with a magnitude of ~200mm on interannual timescales during El Niño. This sea level variability is superimposed on long-term sea level trends and closely related to global temperature evolution and other climate indices. A detailed understanding of climate model's ability in simulating sea level variability and change in the tropical Pacific is crucial for future projections of climate and sea level.

3.29.1 Version & Contact info

- Version/revision information: version 1 (11/16/2020)
- PI (Jianjun Yin, University of Arizona, yin@arizona.edu)
- Developer/point of contact (Chia-Wei Hsu, University of Arizona, chiaweih@arizona.edu)

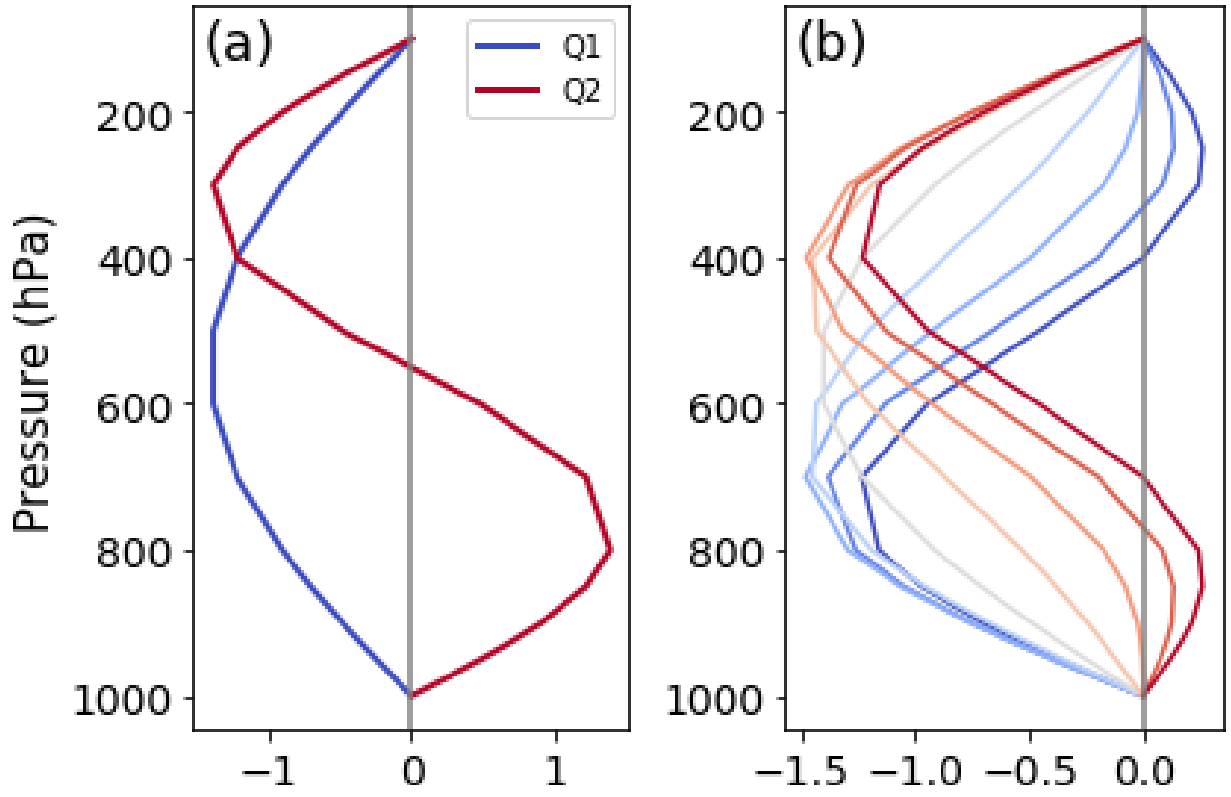


Fig. 14: Figure 1. (a) Q1 and Q2; (b) Vertical velocity profiles constructed from the varying top-heaviness ratio (r; r=-1: dark blue, r=1: dark red).

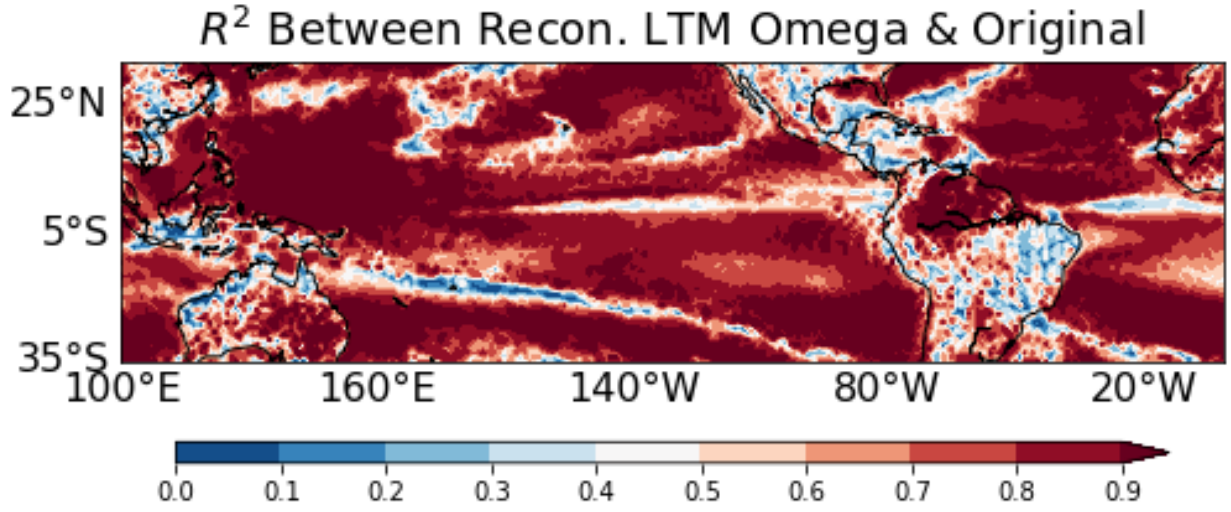


Fig. 15: Figure 2. The fractional variance of explained by ’.

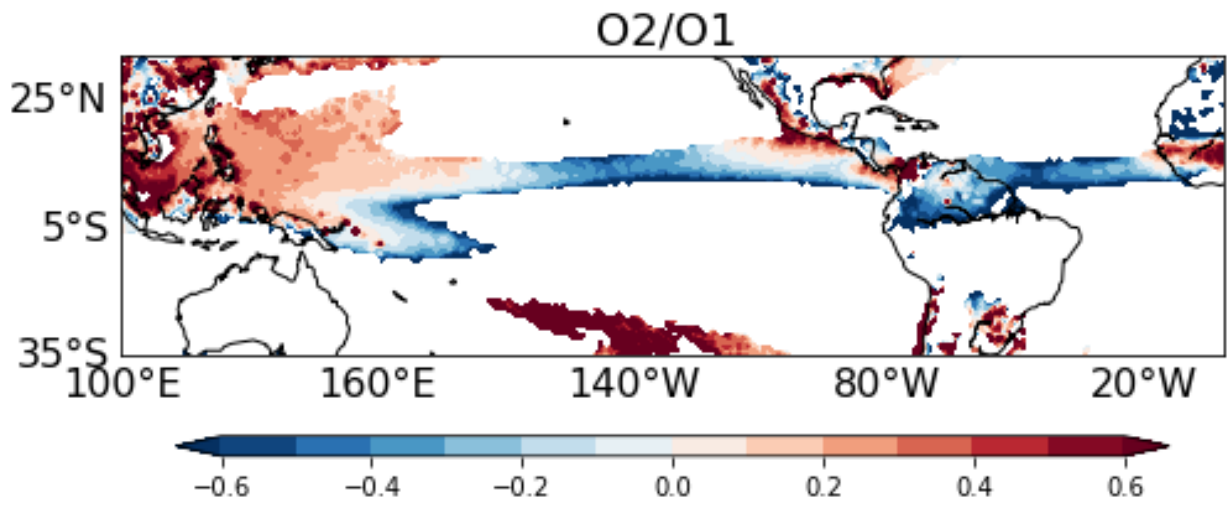


Fig. 16: Figure 3. Long-term mean Top-Heaviness Ratio in July (2000-2019). The ratio is presented only for the grid points with O1 no less than 0.01.

Open source copyright agreement

The MDTF framework is distributed under the LGPLv3 license (see LICENSE.txt). Unless you've distirbuted your script elsewhere, you don't need to change this.

3.29.2 Functionality

The main script generates the tropical Pacific dynamic sea level and wind stress curl scatter plots at different time scales due to their strong dependency from Ekman pumping/suction and barotropic response over the ocean.

Python function used

- spherical_area.cal_area : generate area array based on the lon lat of data
- dynamical_balance2.curl_var_3d : calculate wind stress curl in obs (for Dataset with time dim)
- dynamical_balance2.curl_var : calculate wind stress curl in obs (for Dataset without time dim)
- dynamical_balance2.curl_tau_3d : calculate wind stress curl in model (for Dataset with time dim)
- dynamical_balance2.curl_tau : calculate wind stress curl in model (for Dataset without time dim)
- xr_ufunc.da_linregress : linregress for Dataset with time dim

3.29.3 Required programming language and libraries

The programming language is python version 3 or up. The third-party libraries include “matplotlib”, “xarray”, “cartopy”,”cftime”,”numpy”. The conda environment can be set to _MDTF_python3_base.

3.29.4 Required model output variables

With monthly frequency from the model output. This diagnostic needs

input model variables

- tauuo (surface wind stress in the x direction from native model output resolution/grid)
- tauvo (surface wind stress in the y direction from native model output resolution/grid)
- zos (dynamic sea level height in the model from native model output resolution/grid)

The script is written based on the CESM2-OMIP1 download provided by CMIP6-OMIP hosted by WCRP.

The dimension of all variable is 3-D with (time,nlat,nlon) in dimension and 2-D array for lat and lon as coordinate.

3.29.5 Required observational data

This diagnostic needs

input observational variables

- **adt (absolute dynamic topography from CMEMS)**
  preprocessing from daily to monthly mean is needed (use ‘io_cmems_adt.py’)
- **tx (surface wind stress in the x direction from WASwind)**
  no preprocessing needed
- **ty (surface wind stress in the y direction from WASwind)**
  no preprocessing needed

data access :

- **adt :**
  Ftp server is the fastest way to manage download http://marine.copernicus.eu/services-portfolio/access-to-products/ search for product ID - “SEALEVEL_GLO_PHY_L4_REP_OBSERVATIONS_008_047” Need to download the daily data with adt (absolute dynamic topography) available
- **tx,ty :**
  https://www.riam.kyushu-u.ac.jp/oed/tokinaga/waswind.html

The dimension of all variable is 3-D with 2-D in space and time

3.29.6 References

1. C.-W. Hsu et al. (2020): A Mechanistic Analysis of Tropical Pacific Dynamic Sea Level in GFDL-OM4 under OMIP-I and OMIP-II Forcings. *GMD*, under review.
2. S. M. Griffies et al. (2016): OMIP contribution to CMIP6: experimental and diagnostic protocol for the physical component of the Ocean Model Intercomparison Project. *GMD*, https://doi.org/10.5194/gmd-9-3231-2016
3. S. Kobayashi et al., (2015): The JRA-55 Reanalysis: General Specifications and Basic Characteristics. *Journal of the Meteorological Society of Japan. Ser. II*, https://doi.org/10.2151/jmsj.2015-001
4. W. G. Large and S. G. Yeager, (2009): The global climatology of an interannually varying air–sea flux data set. *Climate Dynamics*, https://doi.org/10.1007/s00382-008-0441-3

3.29.7 More about this diagnostic

The sea level over the tropical Pacific is a key indicator reflecting vertically integrated heat distribution over the ocean. We find persisting mean state dynamic sea level (DSL) bias along 9∘N even with updated wind forcing in JRA55-do relative to CORE. The mean state bias is related to biases in wind stress forcing and geostrophic currents in the 4∘N to 9∘N latitudinal band. The simulation forced by JRA55-do significantly reduces the bias in DSL trend over the northern tropical Pacific relative to CORE. In the CORE forcing, the anomalous westerly wind trend in the eastern tropical Pacific causes an underestimated DSL trend across the entire Pacific basin along 10∘N. The simulation forced by JRA55-do significantly reduces the bias in DSL trend over the northern tropical Pacific relative to CORE. We also identify a bias in the 10 easterly wind trend along 20∘N in both JRA55-do and CORE, thus motivating future improvement. In JRA55-do, an accurate Rossby wave initiated in the eastern tropical Pacific at seasonal time scale corrects a biased seasonal variability of the northern equatorial counter-current in the CORE simulation. Both CORE and JRA55-do generate realistic DSL variation during El Nino. We find an asymmetry in the DSL pattern on two sides of the equator is strongly related to wind stress curl that follows the sea level pressure evolution during El Nino.

3.30 Wavenumber Frequency Spectra Diagnostic Module From NCAR

Last update: 03/11/2019

Produces wavenumber-frequency spectra for OLR, Precipitation, 500hPa Omega, 200hPa wind and 850hPa Wind.

3.30.1 Contact info

- Current Developer: Dani Coleman (bundy@ucar.edu), NCAR
- Contributors: Dennis Shea, Andrew Gettleman, Jack Chen, Rich Neale (NCAR)

Open source copyright agreement

This package is distributed under the LGPLv3 license (see LICENSE.txt).

3.30.2 Functionality

Python code calls NCL wkSpaceTime_driver.ncl code for each of the variables in turn. Preprocessed observational data in the form of gif figures from NCEP precipitation, OLR, Omega and winds, and TRMM precipitation are in the `mdtf/inputdata/obs_data/Wheeler_Kiladis directory` Place your input data at: `mdtf/inputdata/model/$model_name/day` index.html can be found at: `mdtf/MDTF_$ver/wkdir/MDTF_$model_name`

3.30.3 Required Programing Language and libraries

All these scripts required NCAR Command Language Version 6.3.0 or higher

3.30.4 Required input data to the module

Daily U200, U850, OMEGA500, OLR, PRECT

3.30.5 References

1. Wheeler, Matthew, and George N. Kiladis. "Convectively Coupled Equatorial Waves: Analysis of Clouds and Temperature in the Wavenumber–Frequency Domain." *Journal of the Atmospheric Sciences* **56**, no. 3 (February 1, 1999): 374–99. https://doi.org/10.1175/1520-0469(1999)056<0374:CCEWAO>2.0.CO;2.

CHAPTER

FOUR

FRAMEWORK REFERENCE

4.1 Command-line options

4.1.1 Running the package

If you followed the *recommended installation method* (page 4) for installing the framework the conda[173] package manager, the installation process will have created a driver script named `mdtf` in the top level of the code directory. This script should always be used as the entry point for running the package.

This script is minimal and shouldn't conflict with customized shell environments: it only sets the conda environment for the framework and calls mdtf_framework.py[174], the python script which should be used as the entry point if a different installation method was used. In all cases the command-line options are as described here.

4.1.2 Usage

The first form of the command runs the package's diagnostics on model data files in the directory `CASE_ROOT_DIR`. The options, described below, can be set on the command line or in an input file specified with the `-f`` flag. An example of such an input file is provided at src/default_tests.jsonc[175].

The second form of the command prints information about the installed diagnostics. To get a list of topics recognized by the command, run `% mdtf info`.

General options

-h, --help Show a help message, potentially more up-to-date than this page, along with your site's default values

for these options.

-f Path to a user configuration file that sets options listed here. This can be a JSON file of the form given in src/default_tests.jsonc[176] (which is intended to be copied and used as a template)

[173] https://docs.conda.io/en/latest/
[174] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/mdtf_framework.py
[175] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/templates/runtime_config.jsonc
[176] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/templates/runtime_config.jsonc

Path settings

Locations of input and output data. All the paths in this section must be on a locally mounted filesystem. Environment variables in paths (e.g., `$HOME`) are resolved at runtime according to the shell context the package is called from. Relative paths are resolved relative to the code directory.

--OBS-DATA-ROOT <OBS_DATA_ROOT> Required setting if running PODs that require observational data. Directory containing observational and supporting data required by individual PODs. Currently, this must be downloaded manually as part of the framework installation. See Section 1.2.1 of the *installation guide* (page 2) for instructions.

--WORK-DIR <WORKING_DIR> Working directory. This will be used as scratch storage by the framework and the PODs. Optional; defaults to <*OUTPUT_DIR*> if not specified.

-o, --OUTPUT-DIR <OUTPUT_DIR> Required setting. Destination for output files.

Data options

Options that describe the input model data and how it should be obtained.

--convention <naming_convention> The convention for variable names and units used in the input model data. Defaults

to `CMIP`, for data produced as part of CMIP6 data request, or compatible with it.

See the *Recognized conventions* (page 136) for documentation on the recognized values for this option.

--large_file Set this flag when running the package on a large volume of input model data: specifically, if the full

time series for any requested variable is over 4gb. This may impact performance for variables less than 4gb but otherwise has no effect.

When set, this causes the framework and PODs to use the netCDF-4 format (CDF-5 standard, using the HDF5 API; see the netCDF FAQ[177]) for all intermediate data files generated during the package run. If the flag is not set (default), the netCDF4 Classic format is used instead. Regardless of this setting, the package can read

[177] https://www.unidata.ucar.edu/software/netcdf/docs/faq.html#How-many-netCDF-formats-are-there-and-what-are-the-differences-among-them

input model data in
any netCDF4 format.

--disable-preprocessor If set, this flag disables preprocessing of input model data done by the framework before the PODs are run. Specifically, this skips validation of `standard_name` and `units` CF attributes in file metadata, and skips unit conversion and level extraction functions. This is only provided as a workaround for input data which is known to have incorrect metadata: using this flag means that the user assumes responsibility for verifying that the input data has the units requested by all PODs being run.

Conda/micromamba settings

--conda_root path to anaconda, miniconda, or micromamba installation

--conda_env_root path to directory with conda enviroments

--micromamba_exe path to the micromamba executable. REQUIRED if using micromamba

Analysis settings

Settings determining what analyses the package performs.

CASENAME <name> Required setting. Identifier used to label this run of the package. Can be set to any string. startdate <yyyymmdd> or <yyyymmddHHmmss> Required setting. Starting year of analysis period. enddate <yyyymmdd> or <yyyymmddHHmmss> Required setting. Ending year of analysis period. The analysis period is taken to be a **closed interval** pod_list <list of POD identifiers> Specification for which diagnostics (PODs) the package should run on the model data, given as a list separated by spaces. Optional; default behavior is to attempt to run all PODs.

Valid identifiers for PODs are:

- The name of the diagnostic as given in the diagnostics/[178] directory.

Runtime options

Options that control how the package is deployed (how code dependencies are managed) and how the diagnostics are run.

Output options

Options determining what files are output by the package.

save-ps Set flag to have PODs save postscript figures in addition to bitmaps. save-nc Set flag to have PODs save netCDF files of processed data. save-non-nc Set flag to have PODs save all intermediate data **except** netCDF files. make-variab-tar Set flag to save package output in a single .tar file. This will only contain HTML and bitmap plots, regardless of whether the flags above are used. overwrite If this flag is set, new runs of the package will overwrite any pre-existing results in <*OUTPUT_DIR*>.

[178] https://github.com/NOAA/MDTF-diagnostics/tree/main/diagnostics

4.2 Recognized conventions

Naming conventions are specified with the `convention` parameter. The currently implemented naming conventions are:

- `CMIP`: Variable names and units as used in the CMIP6[179]

data request[180]. There is a web interface[181] to the request. Data from any model that ha s been published[182] as part of CMIP6, or processed with the CMOR3[183] tool, should follow this convention.

- `NCAR`: Variable names and units used in the default output of models developed at the

National Center for Atmospheric Research[184] (NCAR), headquartered in Boulder, CO, USA. Recognized synonyms for this convention: `CAM4`, `CESM`, `CESM2`.

- `GFDL`: Variable names and units used in the default output of models developed at the

Geophysical Fluid Dynamics Laboratory[185] (GFDL), Princeton, NJ, USA. Recognized synonyms for this convention: `AM4`, `CM4`, `ESM4`, `SPEAR`.

If you would like the package to support a naming convention that hasn't currently been implemented, please make a request in the appropriate GitHub discussion thread[186].

4.3 Working with unimplemented conventions

The framework has a number of options for handling data that doesn't follow one of the recognized naming conventions described above. All of them involve more manual effort on the part of the user.

- The third-party CMOR[187] tool exists to convert model output into the `CMIP` convention.
- NCO[188], CDO[189] and other utilities provide command-line functionality for renaming variables, unit conversion, editing metadata, etc.
- As mentioned above, `--convention=None` turns off the variable translation functionality. The user is then responsible for ensuring that input model data has the variable names and units expected by each POD.
- Finally, the `--disable-preprocessor` flag skips all unit conversion and checking associated with model metadata. The user is then responsible for ensuring that input model data has the variable names and units expected by each POD.

If using any of the above methods, please carefully consult the documentation for what data is needed by each POD. Note that we do not require POD developers to use any variable naming convention or set of units, so different PODs may request data in mutually inconsistent conventions (e.g., precipitation as a rate vs. as a flux).

[179] https://www.wcrp-climate.org/wgcm-cmip/wgcm-cmip6
[180] https://doi.org/10.5194/gmd-2019-219
[181] http://clipc-services.ceda.ac.uk/dreq/index.html
[182] https://esgf-node.llnl.gov/projects/cmip6/
[183] https://cmor.llnl.gov/
[184] https://ncar.ucar.edu
[185] https://www.gfdl.noaa.gov/
[186] https://github.com/NOAA-GFDL/MDTF-diagnostics/discussions/174
[187] https://cmor.llnl.gov/
[188] http://nco.sourceforge.net/
[189] https://code.mpimet.mpg.de/projects/cdo

4.4 Model data format

This section describes how all input model data must be "formatted" for use by the framework. By "format" we mean not only the binary file format, but also the organization of data within and across files and metadata conventions.

A core design goal of this project is the ability to run diagnostics seamlessly on data from a wide variety of sources, including different formats. The MDTF-diagnostics package leverages ESM-intake catalogs and APIs to query and access the model datasets. As such, we can expand the package requirements to query additional metadata like grid type, institution, or cell methods. If you would like the package to support formats or metadata conventions that aren't currently supported, please make a request in the appropriate GitHub discussion thread[190].

4.4.1 Model data format requirements

File organization

- Model data must be supplied in the form of a set of netCDF or Zarr files with locations and metadata defined in an ESM-intake catalog.
- The framework developers have provided a simple tool for generating data catalogs using CMIP, GFDL, and CESM conventions. The user community may modify this generator to suit their needs
- Each file may contain one variable (i.e., an array with the values of a single dependent variable, along with all of the values of the coordinates at which the dependent variable was sampled), or multiple variables. Refer to the ESM-intake documentation for instructions to create and access data catalogs with multiple assets[191].
- The data for one variable may be spread across multiple netCDF files, but this must take the form of contiguous chunks by date (e.g., one file for 2000-2009, another for 2010-2019, etc.). The spatial coordinates in each file in a series of chunks must be identical.

Coordinates

- The framework currently only supports model data provided on a latitude-longitude grid. The framework developers will extend support for non-rectilinear grids once requirements are finalized and use cases are provided.
- The framework currently only supports vertical coordinates given in terms of pressure. The pressure coordinate may be in any units (*mb*, *Pa*, *atm*, ...). We plan to offer support for parametric vertical coordinates[192] in the near future
- The time coordinate of the data must follow the CF conventions[193]; in particular, it must have a `calendar` attribute which matches one of the CF conventions' recognized calendars (case-insensitive).
- The framework doesn't impose any limitations on the minimum or maximum resolution of model data, beyond the storage and memory available on the machine where the PODs are run.

[190] https://github.com/NOAA-GFDL/MDTF-diagnostics/discussions/174
[191] https://intake-esm.readthedocs.io/en/stable/how-to/use-catalogs-with-assets-containing-multiple-variables.html
[192] http://cfconventions.org/Data/cf-conventions/cf-conventions-1.8/cf-conventions.html#parametric-vertical-coordinate
[193] http://cfconventions.org/Data/cf-conventions/cf-conventions-1.8/cf-conventions.html#time-coordinate

Metadata

The framework currently makes use of the following metadata (attributes for each variable in the netCDF header), in addition to the `calendar` attribute on the time coordinate:

- `units`: Required for all variables and coordinates. This should be a string of the form recognized by UDUNITS2[194], specifically the python cfunits[195] package (which improves CF convention support, e.g. by recognizing `'psu'` as "practical salinity units.")

  This attribute is required because we allow PODs to request model data with specific units, rather than requiring each POD to implement and debug redundant unit conversion logic. Instead, unit checking and conversion is done by the framework. This can't be done if it's not clear what units the input data are in.

- `standard_name`: If present, should be set to a recognized CF convention standard name[196].

  This is used to confirm that the framework has downloaded the physical quantity that the POD has requested, independently of what name the model has given to the variable. If the input files do not contain a *standard_name*, attribute, substitute the *long_name*.

- `realm`: The model realm(s) that each variable is part of.

  If the user or data source has specified a *naming convention* (page 136), missing values for this attribute will be filled in based on the variable names used in that convention.

Many utilities exist for editing metadata in netCDF headers. Popular examples are the ncatted[197] tool in the NCO[198] utilities and the setattribute[199] operator in CDO[200], as well as the functionality provided by xarray itself.

4.4.2 xarray reference implementation

The framework uses xarray[201] to preprocess and validate model data before the PODs are run; specifically using the netcdf4[202] engine and with CF convention support[203] provided via the cftime[204] library. We also use cf_xarray[205] to access data attributes in a more convention-independent way.

If you're deciding how to post-process your model's data for use by the MDTF package, or are debugging issues with your model's data format, it may be simpler to load and examine your data with these packages interactively, rather than by invoking the entire MDTF package. The following python snippet approximates how the framework loads datasets for preprocessing. Use the _MDTF_base[206] conda environment to install the correct versions of each package.

```
import cftime, cf_xarray
import xarray as xr

ds = xr.open_mfdataset(
    [<path to first file>, <second file>, ...],
    parallel=True,
    engine='netcdf4',
```

(continues on next page)

[194] https://www.unidata.ucar.edu/software/udunits/
[195] https://ncas-cms.github.io/cfunits/
[196] http://cfconventions.org/Data/cf-standard-names/77/build/cf-standard-name-table.html
[197] http://nco.sourceforge.net/nco.html#ncatted
[198] http://nco.sourceforge.net/
[199] https://code.mpimet.mpg.de/projects/cdo/embedded/cdo_refcard.pdf
[200] https://code.mpimet.mpg.de/projects/cdo
[201] http://xarray.pydata.org/en/stable/
[202] https://unidata.github.io/netcdf4-python/
[203] http://xarray.pydata.org/en/stable/weather-climate.html#non-standard-calendars-and-dates-outside-the-timestamp-valid-range
[204] https://unidata.github.io/cftime/
[205] https://cf-xarray.readthedocs.io/en/latest/
[206] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/src/conda/env_base.yml

(continued from previous page)

```
    combine='by_coords',
    data_vars='minimal', coords='minimal',
    compat='equals', join='exact',
    decode_cf=True,
    decode_coords=True,
    decode_times=True, use_cftime=True
)
# match coordinates to X/Y/Z/T axes using cf_xarray:
ds = ds.cf.guess_coord_axis()
# print summary
ds.info()
```

The framework has additional logic for cleaning up noncompliant metadata (e.g., stripping whitespace from netCDF headers), but if you can load a dataset with the above commands, the framework should be able to deal with it as well.

If the framework runs into errors when run on a dataset that meets the criteria above, please file a bug report via the GitHub issue tracker[207].

4.5 MDTF-diagnostics Environment variables

This page describes the environment variables that the framework will set for your diagnostic when it's run.

4.5.1 Overview

The MDTF-diagnostics framework can be viewed as a "wrapper" for your code that handles data fetching and munging. Your code communicates with this wrapper in two ways:

- The *settings file* (page 24) is where your code talks to the framework: when you write your code, you document what model data your code uses (not covered on this page, follow the link for details).
- The framework "talks" to a POD through a combination of shell environment variables passed directly to the subprocess via the *env* parameter, and by defining a *case_info.yml* file in the *$WORK_DIR* with case-specific environment variables. The framework communicates **all** runtime information this way: this is in order to 1) pass information in a language-independent way, and 2) to make writing diagnostics easier (i.e., the POD does not need to parse command-line settings).

Note that environment variables are always strings. Your POD will need to cast non-text data to the appropriate type (e.g. the bounds of a case analysis time period, `startdate`, `enddate`, will need to be converted to integers.)

Also note that names of environment variables are case-sensitive.

[207] https://github.com/NOAA-GFDL/MDTF-diagnostics/issues

4.5.2 Paths

The following variables are accessed using the `os.environ` method:

`OBS_DATA`:
Path to the top-level directory containing any observational or reference data you've provided as the author of your diagnostic. Any data your diagnostic uses that doesn't come from the model being analyzed should go here (i.e., you supply it to the framework maintainers, they host it, and the user downloads it when they install the framework). The framework will ensure this is copied to a local filesystem when your diagnostic is run, but this directory should be treated as **read-only**.

`POD_HOME`:
Path to the top-level directory containing your diagnostic's source code. This will be of the form `.../MDTF-diagnostics/diagnostics/<your POD's name>`. This can be used to call sub-scripts from your diagnostic's driver script. This directory should be treated as **read-only**.

`DATA_DIR`:
(retained for backwards compatibility with v3.5 and earlier PODs) location of the model input data directory.

`WORK_DIR`:
Path to your diagnostic's *working directory*, which is where all output data should be written (as well as any temporary files).

The framework creates the following subdirectories within this directory:

- `$WORK_DIR/obs/PS` and `$WORK_DIR/model/PS`: All output plots produced by your diagnostic should be written to one of these two directories. Only files in these locations will be converted to bitmaps for HTML output.
- `$WORK_DIR/obs/netCDF` and `$WORK_DIR/model/netCDF`: Any output data files your diagnostic wants to make available to the user should be saved to one of these two directories.

4.5.3 Model run information

`case_env_file`:
location of the yaml file with case-specific environment variables accessed by calling `os.environ[`case_env_file`]`. The following environment variables are loaded into a dictionary from the case environment file:

`CATALOG_FILE`:
path to the esm-intake catalog header json file used to access the data catalog of processed data files generated by the framework. If `no_pp` is specified at runtime, and no custom preprocessing scripts are run on the input dataset, `CATALOG_FILE` is the path to input data catalog specified with the `DATA_CATALOG` parameter in the runtime configuration file.

`CASENAME`:
User-provided label describing each run of model data being analyzed. Single-run PODs submitted to version 3.5 and earlier of the framework directly access this variable with `os.environ['CASENAME']`.

`startdate`, `enddate`:
Strings in the format <yyyymmdd> or <yyyymmddHHMMSS> describing the start and end dates of the analysis period for a case associated with `CASENAME`. Single-run PODs submitted to version 3.5 and earlier of the framework directly access this variable with `os.environ['startdate]` and `os.environ['enddate]`.

4.5.4 Locations of model data files

The processed model data files are written to the *$WORK_DIR* and accessed via the esm-intake catalog output by the framework, or by the original catalog passed to the framework at runtime if no preprocessing is performed via the `CATALOG_FILE` environment variable in the `case_env_file`

4.5.5 Names of variables and dimensions

These are set depending on the data your diagnostic requests in its *settings file* (page 24). Refer to the examples below if you're unfamiliar with how that file is organized.

4.5.6 Simple example

We only give the relevant parts of the settings file below.

The framework will set the following environment variables in the `case_env_file`:

1. `lat_coord`: Name of the latitude dimension in the model's native format
2. `lon_coord`: Name of the longitude dimension in the model's native format
3. `time_coord`: Name of the time dimension in the model's native format
4. `pr_var`: Name of the precipitation variable
5. `PR_FILE` (retained for backwards compatibility): Absolute path to the file containing `pr` data, e.g. `/dir/precip.nc`.

As with `CASENAME`, `startdate`, and `enddate`, the variable-specific environment variables are accessed with the `os.environ` method in single-run PODs from framework versions older than v4.0.

4.6 Running Submodules

Functions from external packages can be called by MDTF with their inclusion in the json or yml file supplied to the framework.

4.6.1 Inclusion in JSON file

The following block in your JSON or yml file is required for the submodule to launch:

```
"module_list":
 {
   "${MODULE_NAME}":
     {
       "${FUNCTION_NAME}":
         {
           "args":
             {
               "arguments go here"
             }
         }
     }
 },
```

Where, ${MODULE_NAME} is the name for the package you want to launch a function from, ${FUNCTION_NAME} is the function you want to call, and ${FUNCTION_ARGS} is the arguments to be passed to the function.

4.6.2 TempestExtremes Example

As an example, we will build and run TempestExtremes (TE) from MDTF. First, clone the latest TE with a python wrapper. As of writing, this can be found 'here <https://github.com/amberchen122/tempestextremes/>'_ In the cloned directory, it can be built using the commands:

```
python setup.py build_ext
python setup.py install
```

Now, in our JSON file we can call the function DetectNodes by including the following:

```
"module_list":
  {
  "TempestExtremes":
    {
      "DetectNodes":
        {
          "args":
            {
              "--in_data":"./test/cn_files/outCSne30_test2.nc",
              "--timefilter":"6hr",
              "--out":"out1.dat",
              "--searchbymin":"MSL",
              "--closedcontourcmd": "PRMSL_L101,200.,4,0;TMP_L100,-0.4,8.0,1.1",
              "--mergedist":"6.0",
              "--outputcmd": "MSL,min,0;_VECMAG(VAR_10U,VAR_10V),max,2;ZS,min,0",
              "--latname":"lat",
              "--lonname":"lon"
            }
        }
    }
},
```

Multiple packages can be ran if they are nested in "module_list". Multiple functions can be called or even the same one again.

CHAPTER

FIVE

INTERNAL CODE DOCUMENTATION

Warning: The information in this section only pertains to the development and maintenance of the MDTF framework code. It's not needed for end users to run the package, or for POD developers to develop new diagnostics.

5.1 Package code and API documentation

These sections provide an overview of specific parts of the code that's more higher-level than the module docstrings.

5.1.1 Framework configuration and parsing

This section describes the src.cli, responsible for parsing input configuration.

CLI functionality

Overview

Flexibility and extensibility are among the MDTF project's design goals, which must be accommodated by the package's configuration logic. Our use case requires the following features:

- Allow for specifying and recording user input in a file, to allow provenance of package runs and to eliminate the need for long strings of CLI flags.
- Record whether the user has explicitly set an option (to a value which may or may not be the default), or whether the option is unset and its default value is being used.
- Enable site-specific customizations, which can add to or modify any of the above properties.
- Define CLIs through configuration files instead of code to streamline the process of defining all of the above.

The MDTF framework uses the Python Click package[208] to create the CLI from the runtime configuration file options, eliminating the need for custom the CLI modules and plugins in prior versions of the code. Parameters specified in the runtime configuration file (e.g., `templates/runtime_config.[jsonc | yml]`) are passed to a click context object (`ctx`) by attaching the `@click.option` decorator to the `-f` parameter associated with the path to the runtime config file (`e.g., ./mdtf -f [runtime_config_file]`). The ctx object that is instantiated in the *mdtf_framework.py* driver contains the path to the runtime config file passed with the *-f* parameter. *mdtf_framework.py* then calls cli.py utilities to parse the runtime configuration file and pass the the runtime parameters to a *config* dictionary attribute attached to the *ctx* instance. The *config* information is then passed to other framework methods and classes.

[208] https://click.palletsprojects.com/en/8.1.x/

5.1.2 Data layer: Preprocessing

This section describes the src.preprocessor, responsible for converting model data into the format requested by PODs, and the src.xr_parser, responsible for "cleaning" model metadata beforehand. These implement the **Preprocess** stage of the data request

Overview

Functionality

The job of the preprocessor is then to convert the downloaded model data from the model's native format into the format expected by each POD: this is why we use the term "preprocessor," because it operates on model data before the PODs see it.

In full generality, this is a very difficult task: the "format" could refer to any aspect of the model data. Other groups have gone so far as to refer to it as the "holy grail" of portable model analyses, describing it as "CMORization on the fly[209]" (recall that the CMOR[210] tool standardizes model output for CMIP publication; it must be customized for each model convention, and many cases exist where CMIP published data hasn't been perfectly standardized across centers.)

Rather than tackle the full problem at once, we've implemented the preprocessor in a modular way in order to add functionality incrementally, as it's needed by PODs and data sources supported by the package. We break the general "format conversion" problem down into a sequence of individual *transformations* that operate on a single aspect of the data, converting that aspect from what's present in the downloaded data to what's requested by the POD (as described in its settings file and `VarlistEntry` objects). When called, the preprocessor simply executes each transformation in order.

Implementation

Each preprocessor is a class inheriting from `MDTFPreprocessorBase`; a specific child class is associated with each data source via the `_PreprocessorClass` attribute on :class: *~src.data_manager.DataSourceBase* (and all child classes). This lets us handle the case where a specific source of data might require special preprocessing, even though currently all data sources use the `DefaultPreprocessor` class. For example, the methods to open and write the dataset are currently implemented in `DataSourceBase`; a data source that provided model data in Zarr format instead of netCDF would require a new preprocessor class that overrode those methods.

To accomplish the goals above, the preprocessor is structured as a miniature data pipeline. The inputs to the pipeline are the xarray Dataset[211] containing the downloaded data, and the `VarlistEntry` object from the POD describing the requested format for that data.

Methods called

As noted above, the preprocessor has *two* roles: converting the downloaded model data to the format requested by the PODs, and enlarging the scope of the data query to include all formats it's capable of converting between. The latter is executed before the former:

- The preprocessor's `edit_request()` method, called immediately after the preprocessor is initialized.

After this is done, edited data queries are executed using Intake-ESM.

- For every successfully downloaded variable, the `preprocess_data()` method of the data source calls the `process()` method on the POD's preprocessor object that was previously created.

[209] https://docs.esmvaltool.org/en/latest/develop/dataset.html
[210] https://cmor.llnl.gov/
[211] http://xarray.pydata.org/en/stable/generated/xarray.Dataset.html

  - This begins by loading the download variable into an xarray Dataset (`load_ds()`).
  - The process() method on each transformation is called in a fixed order (`process_ds()`).
  - The transformed Dataset is written out to a netCDF file (`write_ds()`).

These aspects are described in more detail below.

Xarray metadata parser

Overview

The job of the metadata parser is to standardize the metadata and other attributes of model data files immediately after they're opened. The goal is for all needed standardization, data validation and other checks to be performed here, so that the logic in the preprocessor transformations can safely make assumptions about the structure of the dataset they operate on, rather than requiring each transformations to code and test for every case it may encounter, which would involve lots of redundant logic.

Like the preprocessor, the parser is implemented as a class so that the functionality can be customized by data sources with different needs, although currently all data sources use the `DefaultDatasetParser`. The preprocessor class to use is specified as the `_PreprocessorClass` attribute on the data source.

Functionality in the parser resists organization, since it needs to be updated to handle every special case of metadata convention encountered in the wild. Broadly speaking, though, the methods are organized into the following stages:

- **Normalize** metadata on the downloaded data: convert equivalent ways to specify a piece of metadata to a single canonical representation.
- **Reconcile** the metadata with what the POD expects. Recall that each VarlistEntry is converted to a `TranslatedVarlistEntry`, expressing the variable in the model's native convention. In this stage, we check that the variable we *expected* to download, as expressed in the TranslatedVarlistEntry, matches what was *actually* downloaded. If there are differences, we update either the data's metadata or the TranslatedVarlistEntry, or raise an error.
- **Check** metadata admissibility before exiting, raising errors if necessary. It's conceptually simpler to write these tests as a separate stage that covers everything than to integrate the tests piecemeal into the previous two stages.

Method names in the parser follow this convention.

Methods called

The parser has one public method, `parse()`, which is the entry point for all functionality. It calls the following methods:

- `normalize_pre_decode()` strips leading/trailing whitespace and does other proofreading on the raw xarray attributes. It also makes a copy of the raw attributes, since they can be overwritten by the next two methods.
- xarray's[212] own decode_cf()[213] method, which primarily decodes

  the time coordinate to cftime.datetime[214] objects, which are properly calendar-aware.
- cf_xarray's[215] guess_coord_axis()[216]

[212] http://xarray.pydata.org/en/stable/index.html
[213] http://xarray.pydata.org/en/stable/generated/xarray.decode_cf.html
[214] https://unidata.github.io/cftime/api.html#cftime.datetime
[215] https://cf-xarray.readthedocs.io/en/latest/index.html
[216] https://cf-xarray.readthedocs.io/en/latest/generated/xarray.DataArray.cf.guess_coord_axis.html#xarray.DataArray.cf.guess_coord_axis

method, which uses heuristics to assign axis labels ('X', 'Y', 'Z', 'T') to dataset coordinates. This is important, since we need a way to handle the data's coordinates that doesn't depend on the model's naming conventions and coordinate system.

- `restore_attrs_backup()` corrects any metadata that was overwritten.
- `normalize_metadata()` then does our own normalization:
  - For all variables (dependent variables and coordinates) in the dataset, we normalize the standard name (`normalize_standard_name()`) and units attributes (`normalize_unit()`).
  - `normalize_dependent_var()` verifies that a dependent variable exists in the dataset matching the name expected in the TranslatedVarlistEntry.
- `check_calendar()` checks whether decode_cf() parsed the date axis correctly, and if not, looks for calendar information in some non-standard locations. This is needed before we do reconciliation tasks involving the time coordinate.
- `reconcile_variable()` then reconciles the data's metadata with the expected metadata from the TranslatedVarlistEntry. In general, missing metadata from either source is filled in with values from the other source, while explicit differences in metadata attributes raise an error.
  - `reconcile_names()` reconciles the variable's name and its standard name attribute.
  - `reconcile_units()` reconciles the units attribute. An error is raised if the units are not equivalent, but unequal units are OK.
  - `reconcile_dimension_coords()` does similar logic for the variable's dimension coordinates, also reconciling the coordinate's bounds variable if present.
  - `reconcile_scalar_coords()` does similar logic for the variable's scalar coordinates (levels of a 3D variable.)
- `check_ds_attrs()` does all remaining checks on the final state of the metadata:
  - We verify the calendar is still set correctly.
  - For all variables, we ensure that valid standard name and units attributes were assigned.

At this point, the metadata on the dataset is ready for use by the preprocessor's transformations.

Intake-ESM

We use intake ESM to load and manipulate all model data, as it's

Preprocessor functions

Overview

As described above, preprocessor transformations aren't implemented as simple python functions, because they have two roles: to actually perform the conversion, and to expand the scope of the data query to include all data formats they can convert between. Because of this, transformations are implemented as classes with two methods for the two roles: `edit_request()` and `process()`. The abstract base class defining these is `PreprocessorFunctionBase`. (Replacing "Function" with "Transformation" in the class names would be less confusing.)

Editing the data request

Recall that by "data request," we mean the linked list of VarlistEntry objects connected through the `alternates` attribute. The **Query** stage of the data source traverses this list in breadth-first order until a viable set of alternates is found: if the data specified by one VarlistEntry isn't available, we try its alternates (if it has any), and if one of those isn't found, we try its alternates, and so on. "Editing the data request" corresponds to inserting new VarlistEntry objects into this linked list corresponding to the alternatives we want to consider.

Some transformations don't need to implement edit_request(). For example, `ConvertUnitsFunction`: units are uniquely determined by the variable name and model's variable convention; no data source saves multiple copies of the same variable in different units.

An simple example of a transformation that implements edit_request() is `PrecipRateToFluxFunction`: different models and different PODs define precipitation as a rate or as a mass flux. It's easy to convert between the two, but because it falls outside the scope of the udunits2 library we handle it as a special case here.

A POD that needs precipitation will request it as either a rate or a flux, but because we can convert between the two, we should also add the other quantity as an alternate variable to query. This is done by the `edit_request()` method: it takes a VarlistEntry *v* and, if it refers to precipitation rate or flux, returns an edited copy *new_v* referring to the other quantity (and returning None otherwise.) The decorator `edit_request_wrapper()` then does the bookkeeping work of inserting *new_v* after *v* in the linked list of alternate variables for the POD – because this is the expected scenario for editing the data request, we collect the logic in one place.

Provenance

Log messages with the ObjectLogTag.NC_HISTORY tag will be copied to the `history` attribute of the netCDF file written as the output of the preprocessor, in case the user wishes to use these files for a non-MDTF purpose. In general, preprocessor transformations should be verbose in logging, since this section of the code is key to diagnosing problems arising from malformed model data.

5.1.3 util subpackage

This section summarizes code in the `src/util` subpackage, which contains utility functions needed at many places in the code. It's implemented as a package because putting all the code in a single module would be difficult to navigate. All modules depend on the python standard library only.

This section is intended to give an introduction and context for the overall code organization, which might be difficult to gain from the complete docstring listing at the *Internal code documentation* (page 143). In particular, we don't describe every class or function in detail here.

Modules in the package

__init__.py

The util package contains a non-trivial `__init__.py`, describing the "public" members of each module that are provided when the `util` package is imported as a whole. New classes or functions added to a module in the package should be listed here in order to be usable outside the package.

src.util.basic

This module contains implementations of the simplest data structures and utility functions needed by the framework. In general, code is placed in this module to avoid the need for circular imports (which can be done safely in python, but which we avoid.)

src.util.dataclass

This module contains extensions to the python standard library's `dataclasses`[217] and `re`[218] modules. Recall that python dataclasses provide an alternative syntax for class definition that's most useful in defining "passive" classes that mainly serve as a container for typed data fields (hence the name), for example, records of a database. The code in this module extends this functionality to the use case of instantiating dataclasses from the results of a regex match on a string: *our* main use case is assembling a data catalog on the fly by using a regex to parse paths of model data files in a set directory hierarchy convention.

For a simple example of how the major functionality of this module is used in the framework, examine the code for `src.cmip6.CMIP6_VariantLabel`. This class simply parses a variant label of the type used in the CMIP6 conventions: given a string of the form `r1i2f3`, return a CMIP6_VariantLabel object with `realization_index` attribute set to `1`, etc.

Regexes

The `RegexPattern` class extends the functionality of the standard library's `re.Pattern`[219]: it wraps a regex where all capture groups are named, and provides a dict-like interface to the values captured by those groups upon a successful match.

A `ChainedRegexPattern` is instantiated from multiple `RegexPattern`s: given an input string, it tries parsing it with each RegexPattern in order, stopping at the first one that matches successfully. In other words, this is a convenience wrapper for taking the logical 'or' of the RegexPatterns, which is cumbersome to do at the level of the regexes themselves.

Dataclasses

We implement the `src.util.dataclass.mdtf_dataclass()` decorator to smooth over the following rough edges in the standard library implementation of dataclasses – otherwise, usage is unchanged from `dataclasses.dataclass()`[220].

- Re-implement checking for mandatory fields. Standard library dataclasses allow for both mandatory and optional fields, but the optional fields must be declared after the mandatory ones, which breaks when dataclasses inherit from other dataclasses (the parent class's fields are declared first in the auto-generated `__init__` method).

  Our workaround is to always declare fields as optional (in the context of the standard library's dataclass, that we're wrapping) and denote those that are meant to be mandatory with a default sentinel value.

- Perform type coercion on instance creation (after the class's `__init__` and `__post_init__`). Python is committed to being a weakly ("duck") typed language, which won't do for our use case: the field values returned by the regex will all be strings, and we want to coerce these to ints, dates, etc. using the pre-existing dataclass type annotation syntax.

[217] https://docs.python.org/3.11/library/dataclasses.html#module-dataclasses
[218] https://docs.python.org/3.11/library/re.html#module-re
[219] https://docs.python.org/3.11/library/re.html#re.Pattern
[220] https://docs.python.org/3.11/library/dataclasses.html#dataclasses.dataclass

The logic for doing so is in `_mdtf_dataclass_typecheck()`: implementing full type awareness (as done by `mypy` or similar projects) is far beyond our scope, so this only does coercion on the simplest cases that actually arise in practice and throws a `DataclassParseError` if it encounters anything it can't understand.

"Regex dataclasses"

The regex and dataclass functionalities described above are combined using the `regex_dataclass()` decorator. Its argument is a RegexPattern instance, and it decorates a mdtf_dataclass, and its main function is to wrap the auto-generated `__init__` method to allow the mdtf_dataclass to be instantiated from parsing a string using the RegexPattern.

Extra effort is needed to make this work properly under composition (i.e., if the types of one or more of the fields of the current regex_dataclass are *also* regex_dataclasses.) This is mainly done in `_regex_dataclass_preprocess_kwargs()`: we parse the constituent regex_dataclasses in depth-first order, and keep track of their field assignments in a `ConsistentDict` which throws an exception if we try to alter a previously defined value.

Other functionality

Interoperability between standard library dataclasses is cumbersome: e.g. if a dataclass has a field named `id`, there's no straightforward way to relate it to the `id` field on a different class, even if one inherits from the other. We implement two functions for this purpose, which are roughly inverses of each other.

`filter_dataclass()` returns a dict of the field values in one dataclass that correspond to fields names that are present in a second dataclass. `coerce_to_dataclass()` creates an instance of a given dataclass using field values specified by a second dataclass, or a dict.

src.util.datelabel

This module implements classes for representing the date range of data sets and the frequency with which they are sampled. As the warnings on the module's docstring should make clear, this is **not** intended to provide a full implementation of calendar math. The intended use case is parsing date ranges given as parts of filenames (hence "datelabel") for the purpose of determining whether that data falls within the analysis period.

Date ranges and dates

Date ranges are described by the `DateRange` class. This stores the two endpoints of the date range as `datetime.datetime`[221] objects, as well as a precision attribute specified by the `DatePrecision` enum. DateRanges are always **closed** intervals; e.g. `DateRange('1990-1999')` starts at 0:00 on 1 Jan 1990 and ends at 23:59 on 31 Dec 1999. In all cases, the DateRange is defined to be the maximal range of dates consistent with the input string (i.e., the precision with which that string was specified).

Because we retain precision information, the `Date` class is implemented as a DateRange, rather than the other way around; for example `DateRange('1990')` has yearly precision, so it maps to the range of dates from 0:00 on 1 Jan 1990 to 23:59 on 31 Dec 1990.

[221] https://docs.python.org/3.11/library/datetime.html#datetime.datetime

Sampling frequencies

The frequency with which data is sampled is represented by the `DateFrequency` class, which is essentially a wrapper for the standard library's `datetime.timedelta`[222] that provides string parsing logic.

Static data

The module defines `FXDateRange`, `FXDateMin`, `FXDateMax` and `FXDateFrequency` placeholder objects to describe static data with no time dependence. These are defined at the module level, so they behave like singletons. Comparisons and logic with normal DateRange, Date and DateFrequency objects work correctly.

src.util.exceptions

In order to simplify the set of modules imported by other framework modules, all framework-specific exceptions are defined in this module, regardless of context. All framework-specific exceptions inherit from `MDTFBaseException`.

src.util.filesystem

Functionality that touches the filesystem: path operations, searching for and loading files (note that the parsing of files is done elsewhere), and (simple) HTML templating for the src.output_manager.

src.util.logs

Functionality involving logging configuration and output. Code in this module extends the functionality of the python standard library `logging`[223] module, which we use for all user communication during framework operation (instead of `print()` statements). Python's built-in logging facilities are powerful, going most of the way towards implementing an event-driven programming paradigm within the language, and not very clearly documented. The tutorial[224] is a must-read.

Configuration

In keeping with the framework's philosophy of extensibility, we want to allow the user to configure logging themselves (e.g., they may want errors raised by the MDTF package to be reported to a larger workflow engine.) We do this by simply exposing the logging module's configuration interface[225] to the user: specifically, the `dictConfig()`[226] schema[227], with the contents of the dict serialized as a .jsonc file. We do this rather than using the `fileConfig()`[228] interface, because the latter uses files in .ini format, and we currently use .jsonc for all other configuration files in the package.

Specifically, the framework looks for logging configuration in a file named `logging.jsonc`, as part of the `MDTFFramework`'s `__init__` method. It first looks in the `site` directory specified by the user; if no file with that name is found, it falls back to the default configuration in src/logging.jsonc[229]. The contents of this file are stored

[222] https://docs.python.org/3.11/library/datetime.html#datetime.timedelta
[223] https://docs.python.org/3.11/library/logging.html#module-logging
[224] https://docs.python.org/3.7/howto/logging.html#logging-basic-tutorial
[225] https://docs.python.org/3.7/library/logging.config.html
[226] https://docs.python.org/3.11/library/logging.config.html#logging.config.dictConfig
[227] https://docs.python.org/3.7/library/logging.config.html#logging-config-dictschema
[228] https://docs.python.org/3.11/library/logging.config.html#logging.config.fileConfig
[229] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/src/logging.jsonc

in the ConfigManager and actually used to configure the logger by case_log_config(), which gets called by the __init__ method of DataSourceBase.

Caching

The configuration strategy described above creates a chicken-and-egg problem, as we need to be able to log issues that arise before the logger itself has been configured. We do this with the MultiFlushMemoryHandler log handler, which acts as a temporary cache: all logging events prior to configuration are captured by this handler. Once the "real" handlers have been configured by case_log_config(), the contents of this handler are copied ("flushed") to each of them in turn. This handler is set up in the top-level script, which also calls configure_console_loggers() to set up conventional stdout/stderr logging destinations.

Most of the rest of the code in this module deals with formatting and presentation of logs, e.g. MDTFHeaderFileHandler which writes a header with useful debugging information (such as the git commit hash) to the log file.

src.util.processes

Functionality that involves external subprocesses spawned by the framework. This is the mechanism by which the framework calls all external executables, e.g. tar. We implement two main functions which take the same arguments: run_shell_command(), for running commands in a shell environment (e.g. permitting the use of environment variables), and run_command(), for spawning a subprocess with the executable directly, without the overhead of starting up a shell. Both of these are effectively convenience wrappers around the python standard library's subprocess.Popen[230].

Note that, due to implementation reasons, SubprocessRuntimeManager doesn't call run_shell_command() but instead implements its own wrapper.

5.2 Module index

5.2.1 Main framework modules

5.2.2 Supporting framework modules

[230] https://docs.python.org/3.11/library/subprocess.html#subprocess.Popen

5.2.3 Utility modules

The `src.util` subpackage provides non-MDTF-specific utility functionality used many places in the modules above. See the *util subpackage* (page 147) documentation for an overview.

`src.util.basic`	Classes and functions that define and operate on basic data structures.
`src.util.dataclass`	Extensions to Python `dataclasses`[231], for streamlined class definition.
`src.util.datelabel`	Classes for serializing and deserializing dates and times expressed as strings in filenames and paths.
`src.util.exceptions`	All framework-specific exceptions are placed in a single module to simplify imports.
`src.util.filesystem`	Utility functions for interacting with the local filesystem and configuration files.
`src.util.logs`	Utilities related to configuration and handling of framework logging.
`src.util.path_utils`	Utility functions for defining directory paths
`src.util.processes`	Utility functions for dealing with subprocesses.

[231] https://docs.python.org/3.11/library/dataclasses.html#module-dataclasses

CHAPTER

SIX

TOOLS DOCUMENTATION

6.1 catalog_builder.py

6.1.1 USAGE

Generate ESM-intake catalogs for datasets stored using the CMIP6, CESM, and GFDL archive directory and retrieval structures (DRSs).

To run interactively:

```
> cd MDTF-diagnostics/tools/catalog_builder
> conda activate _MDTF_base
> python3 catalog_builder.py --config [CONFIG FILE NAME].yml
```

Submit a SLURM batch job:

```
> sbatch catalog_builder_slurm.csh -config [CONFIG FILE NAME].yml
```

6.1.2 Input

Yaml file with configuration to build an ESM intake catalog

6.1.3 Output

A csv file with ESM-intake catalog entries for the target root directory(ies) in the configuration file, and a json file with the catalog column headers. Example catalog and header files for CMIP6 dataset stored on the GFDL uda file system are located in the examples/cmip subdirectory.

6.1.4 Required packages:

The required packages are included in the _MDTF_base conda environment:

- click
- dask
- datetime
- ecgtools
- intake

- os
- pathlib
- shutil
- sys
- time
- traceback
- typing
- xarray
- yaml

6.1.5 Configuration file:

The configuration file defines the following parameters to generate the ESM-intake catalog:

- convention (required): DRS convention to use: cmip (default), gfdl, or cesm
- data_root_dirs (required): a list of root directory paths with files to query
- dir_depth (required): the directory depth to traverse in the paths. A dir_depth=1 means that the files are in the root directory(ies), a dir_depth=2 means the files are in one or more subdirectories one level down from the root directory(ies) and so on
- output_dir (required): directory where catalog and header files will be written
- output_filename (required): base name of the catalog and header files (.csv and .json are appended by the program)
- num_threads (required): number of cpu threads to run with
- include_patterns (optional): list of patterns to include in search; supports wildcards
- exclude_patterns (optional): list of patterns to exclude from search; supports wildcards

Templates for the configuration file and a slurm batch submission script for GFDL PPAN are located in the examples/templates subdirectory.

6.2 rename_input_files.py

6.2.1 USAGE

Rename input files that do not adhere to the default Local_file data_manager convention of *<CASENAME>.<frequency>.<variable name>.nc* and write to the directory *[outputDir]/[CASENAME]/[frequency]*

To use, run the following command:

> ./rename_input_files.py –config_file [configuration file name].py

6.2.2 Input

Configuration yaml file with the directory containing the input data, the directory where the output will be written, the CASENAME to use for the output file names, the file names for the copied data, and the frequencies and variable names to use in the new file names

The file *config_template.yml* shows how to define the *casename* and *frequency* parameters, the paths to the original input data and the output directory where the modified file names will be copied, the names of the files to change, and the corresponding variable names that will be substituted in the modfied file names.

6.2.3 Output

Copies of the desired files to the directory *[outputDir]/[CASENAME]/[freq]* with the format *<CASENAME>.<freq>.<variable name>.nc*

6.2.4 Required packages:

The required packages are included in the _MDTF_base conda environment: - os - sys - pyyaml - click - pathlib - shutil