Getting started with the MDTF package

Yi-Hung Kuo (UCLA), Dani Coleman (NCAR). Last update: 6/26/2019

This document provides directions for downloading, installing and running a test of the Model Diagnostics Task Force (MDTF) Process-Oriented Diagnostics package using sample model data.

For descriptions of the Process-Oriented Diagnostic modules (referred to as PODs hereafter) included in the package, and sample output, visit the MDTF main page. The package is provided as-is. Questions about individual PODs should go to the contributing groups.

For developers contributing a POD to the package, see Developers' Walkthrough.

Summary of steps for running the package:

I. Download (a) code package, (b) pre-digested observational data, and (c) two sets of sample model data. See section 1.
II. Install (a) NCL and (b) Python with libraries. See section 2.
III. Run the package by executing *python mdtf.py namelist* in a terminal. See section 3.

Below, sections 1-4 summarize steps for testing the package with provided samples. For advanced users, sections 5-6 detail how to run the package with your own model data. Section 7 is a very short list for troubleshooting.

1. Download the package code and sample data for testing

Download the MDTF code package, pre-digested observational data, and two sets of sample model data from the links in the Downloading and Running section on the MDTF main page. After downloading, untar the files:

```
% tar xf MDTF_$ver.var_code.tar[1]
% tar xf MDTF_$ver.obs_data.tar
% tar xf model.QBOi.EXP1.AMIP.001.tar
% tar xf model.GFDL.CM4.c96L32.am4g10r8.tar
```

This creates a directory named *mdtf* containing sub-directories for the code package and input data. Execute:

```
% ls mdtf
```

the following directories and files under *mdtf* will appear:

```
MDTF_$ver:
   LICENSE.txt      mdtf.py          namelist         var_code/
inputdata:
   model/           obs_data/
```

[1] Throughout this document, % indicates the UNIX/LINUX command line prompt and is followed by command to be executed in a terminal, and $ indicates strings to be substituted, e.g., the string *$ver* here should be substituted by the actual version in the tar file name.

make_tars.sh

Here, the main script *mdtf.py* will call POD scripts under *var_code* following the settings specified in *namelist*. The observational data is stored under *obs_data*. The sample model data, one from a particular run of the NCAR CESM (CASENAME: QBOi.EXP1.AMIP.001), and another from a GFDL CM4 run (CASENAME: GFDL.CM4.c96L32.am4g10r8), are under *model*.

The default test case uses the QBOi.EXP1.AMIP.001 sample. The GFDL.CM4.c96L32.am4g10r8 sample is only for testing the MJO Propagation and Amplitude POD, see section 3. For instructions on using your own data set, see section 5.

2. Install the necessary programming languages and modules

The MDTF package is primarily written in Python, with some of its PODs written in NCL. Thus, running the complete MDTF package requires installing both languages.

1) NCL installation

On a Linux machine, we recommend the users to install NCL (version 6.4.0; tested) by downloading the precompiled binaries through the link:

https://www.earthsystemgrid.org/dataset/ncl.640.dap/file.html

A newer version 6.5.0 is also available (change *640* in the above link to *650*). Choose a download option according to the Linux distribution and hardware, unzip the file (results in 3 folders: *bin*, *include*, *lib*), create a folder *ncl* under the directory */usr/local* (requires permission) and move the 3 unzipped folders into */usr/local/ncl*. Then add the following lines to the *.bashrc* script (under the user's home directory; may be different if using shells other than bash, e.g., *.cshrc* for csh):

```
export NCARG_ROOT=/usr/local/ncl
export PATH:$NCARG_ROOT/bin:$PATH
```

2) Python installation

The MDTF package requires Python (currently tested for version 2.7; update to 3 is planned to be done after this release) and additional modules (e.g., *netcdf4*, *networkx*, and *numba*) that may not be included in the standard Python installation. We thus recommend that users use Anaconda[2] – a non-proprietary Python package manager – to manage the Python modules.

The Anaconda 2 installer, version 5.0.1, available at https://repo.continuum.io/archive/, has been tested for the MDTF package. Since some of the MDTF package plotting scripts are sensitive to the *matplotlib* module whose version changes with the installer, we recommend that users use this version. The installer, by default, will install Anaconda, Python and common modules in the users' home directory. Toward the end of the installation process, the installer will add the following line to *.bashrc* (requires the users' consent):

```
export PATH="home/user/anaconda2/bin:$PATH"
```

[2] Official website: https://www.anaconda.com. The *networkx* and *numba* modules will be included in the default Anaconda installation.

Subsequently, the additional Python modules, e.g., *netcdf4*, required for the MDTF package can be installed by executing the following line in the terminal:

```
% conda install netcdf4
```

3. Execute the MDTF package

Go to the *mdtf/MDTF_$ver* directory. The main script *mdtf.py* therein reads all input settings from a text file *namelist* given as an argument on the command line:

```
% cd mdtf/MDTF_$ver
% python mdtf.py namelist[3]
```

Running the test case set by the default *namelist* file will take up to 8GB of RAM and ~30 min. This default case will execute through all the available PODs except for the Diurnal Cycle of Precipitation POD (and MJO Propagation and Amplitude POD which uses the GFDL.CM4.c96L32.am4g10r8 sample).

To run the Diurnal Cycle of Precipitation POD (tagged as *precip_diurnal_cycle* in the code), edit the *namelist* by deleting the # sign at start of the line *#POD precip_diurnal_cycle*, then execute the *mdtf.py* script again as instructed above. This will take up to 16GB of RAM and another ~10 min. Conversely, to skip any PODs from the default, simply comment out the *POD* lines with a # sign.

To test the MJO Propagation and Amplitude POD, use the other namelist file *namelist_mjo_pro* by executing *python mdtf.py namelist_mjo_pro*. Comparing the two available namelist examples, the latter one has a different CASE statement *CASE GFDL.CM4.c96L32.am4g10r8 AM4 1977 1981* and an additional *POD MJO_prop_amp* line with all the other POD lines commented out.

4. View the results

The package writes output into *mdtf/MDTF_$vers/wkdir/MDTF_QBOi.EXP1.AMIP.001*[4]. Each POD writes a webpage (i.e., an *html* file) to link to the figures and their observational counterparts.

The users can view the results by simply using a web browser (e.g., Firefox/Chrome/Safari) to open *mdtf/MDTF_$vers/wkdir/MDTF_QBOi.EXP1.AMIP.001/index.html*, and then clicking through the links.

In case the code package is run on a sever without GUI, the users can copy the tarball of *wkdir* (.tar; created by default) to a local machine using the *scp* command, and view the results through a browser:

```
% scp mdtf/MDTF_$vers/wkdir/MDTF_QBOi.EXP1.AMIP.001.tar $localmachine:$dir[5]
login to the local machine
% cd $dir
% tar xf MDTF_QBOi.EXP1.AMIP.001.tar
view MDTF_QBOi.EXP1.AMIP.001/index.html using a browser
```

The results should match this sample webpage output of the package.

---

[3] Or *python mdtf.py namelist >> mdtf.log* to capture output in log file *mdtf.log*.
[4] The name of the directory and the tarball file mentioned below change with the *CASENAME* set in *namelist*.
[5] Here, *$localmachine* and *$dir* are to be substituted with the actual local machine address and directory for saving the tarball.

5. Using your own model data

To run the diagnostics on your own model data, the following steps are explained below:

1. Place model data in the expected directory structure with the expected filenames.
2. Create a namelist file for the MDTF package specifying your case name and time bounds.

1) Place model data in the expected directory structure with the expected filenames

One can refer to the sample model data directories and filenames as an example. The model data to be analyzed should be stored in the directory *mdtf/inputdata/model/CASENAME*, under which the time-frequency sub-directories (*1hr*, *3hr*, *day*, *mon*) can be found. The files should be in the NetCDF format with the specific filename structure:

CASENAME.VARNAME.FREQ.nc

where

CASENAME = the name of the model experiment (must match the directory under *mdtf/inputdata/model/*)
VARNAME = the variable name as specified in *mdtf/MDTF_$ver/var_code/util/set_variables_$model.py*[6]
FREQ = the time frequency (must match the name of its directory)

For example, using *f.e.20* as the casename, with monthly mean data already in *$mydir* written in CESM timeseries files:

```
% cd mdtf/inputdata/model
% mkdir f.e.20
% cd f.e.20
% mkdir 1hr 3hr day mon
% cd mon
% ln -s $mydir/atm/proc/tseries/month_1/f.e.20.cam.h0.PRECT.000101-000512.nc \
        f.e.20.PREC.mon.nc
```

The last line creates a link *f.e.20.PREC.mon.nc* under *mdtf/inputdata/model/f.e.20/mon* to an existing file *f.e.20.cam.h0.PRECT.000101-000512.nc*.

2) Create a namelist file for the MDTF package specifying your case name and time bounds

The default settings for the code package are given in *mdtf/MDTF_$ver/namelist*. You can modify this file directly or copy it to a new file and give that as an argument to *mdtf.py*.

The case information is specified in *namelist* by names with *CASE* at the front of them in the following format:

CASE CASENAME model FIRSTYR LASTYR

[6] Here, *$model* is to be substituted with the actual name of the model, e.g., CESM or AM4. The script *set_variables_CESM.py* specifies variable names following the CESM naming convention. There is also an example script *set_variables_CMIP.py* following the CMIP convention.

For example, using our previous case:

% cp namelist namelist_f.e.20
edit CASE line in namelist_f.e.20 to read: *CASE f.e.20 CESM 2001 2005*
% python mdtf.py namelist_f.e.20

Please note that the code package is currently only capable of running one case at a time. See section 6 for more regarding the namelist files.

6. Modifying package settings

Namelist file usage

All settings should be made in the namelist file given as an argument to *mdtf.py*. If no argument is given, *mdtf.py* will look for a file called *namelist*, but users can use other filenames for multiple invocations, e.g.,

% python mdtf.py namelist_f.e.20

Namelist file example

```
# Comment out anything with # at start of line
# CASE CASENAME model FIRSTYR LASTYR
CASE QBOi.EXP1.AMIP.001 CESM 1977 1981

# Packages are specified with POD tag at the start of the line
POD Wheeler_Kiladis
POD EOF_500hPa
POD precip_diurnal_cycle
POD convective_transition_diag
POD MJO_suite
POD MJO_teleconnection
#POD Something-else-we-don't-want-to-run-so-we-commented-it-out

# List any envvars to be set here with VAR tag at the start of the line
# These will override any settings in mdtf.py
VAR make_variab_tar 1
VAR verbose 1 # 0-minimal,1-normal,2-copious,3-debug
#VAR NCARG_ROOT "/glade/u/apps/ch/opt/ncl/6.4.0/intel/17.0.1"   #ncar only
VAR CLEAN  0 # don't remove existing files
VAR test_mode False #True = script just reports what it would do, doesn't call actual packages
```

Namelist file overview

The namelist file has 3 types of entries, tagged by the first word in the line: *CASE*, *POD*, or *VAR*.

1) Namelist file entry: CASE

Lines that start with CASE specify a model experiment. Note that, although the design allows the user to enter more than one case, the package does no yet run more than one. The format is strict:

CASE CASENAME model FIRSTYR LASTYR

- *CASENAME* must exactly match the directory in *mdtf/inputdata/model*

- *model* = CMIP or CESM (or other model type), determines variable names as set in *mdtf/MDTF_$ver/var_code/util/set_variables_$model.py*
- *FIRSTYR* and *LASTYR* specify the time period to be analyzed (only used by some PODs)

e.g.,

```
CASE QBOi.EXP1.AMIP001 CESM 1977 1981
```

2) Namelist file entry: POD

Lines that start with *POD* choose Process-Oriented Diagnostic modules to run. They expect the name of a diagnostic module, which must exactly match the name of the directory in *mdtf/MDTF_$ver/var_code*. There can be an unlimited number of modules requested and *POD* lines can be commented out with # to quickly disable their execution.

For example:

```
POD Wheeler_Kiladis
POD EOF_500hPa
POD convective_transition_diag
#POD fortran_example          #this will not be run
POD MJO_suite
```

3) Namelist file entry: VAR

Lines that start with *VAR* set environment variables used by *mdtf.py* and any of the packages (see Developers' Walkthrough for a complete list of environment variables). They expect 2 arguments with the format:

```
VAR var-name var-value
```

For example:

```
VAR make_variab_tar 1 # make a tar file of wkdir when complete
VAR verbose         1 # 0-minimal,1-normal,2-copious,3-debug
VAR CLEAN           0 # don't remove existing files
VAR test_mode       True #True = reports what it would do, False = executes packages
VAR NCARG_ROOT      "/glade/u/apps/ch/opt/ncl/6.4.0/intel/17.0.1" #ncar only
```

These are set as shell environment variables in the shell and available for the PODs to use. Note that namelist var settings override any environ settings in *mdtf.py*, so the user can add lines to.

Required namelist settings

mdtf.py will fail if the required case variables, and any required environment variables (currently only *NCARG_ROOT*, which sets the path for NCL) fail. If *NCARG_ROOT* is set in the environment [section 2, 1)], it does not need to be set in the namelist.

Namelist archival

A namelist file is written with every execution of *mdtf.py* in order to assist with reproducibility. This is archived in *mdtf/MDTF_$ver/MDTF_$CASENAME/namelist.YYYYMMDDHHMM*.

7. Troubleshooting

Here we provide a short list of problems the MDTF team had previously encountered.

1) The error message "*convert: not authorized …*" shows up:

The MDTF package generates figures in the PostScript (PS) format, and then uses the *convert* command (from the ImageMagick software suite) to convert the PS files to PNG files. The *convert* error can occur after recent updates and can be solved as follows (requires permission):

In the file */etc/ImageMagick/policy.xml*,[7] change line

<policy domain="coder" rights="none" pattern="PS" />

to

<policy domain="coder" rights="read|write" pattern="PS" />

2) Convective transition diagnostic module is not executed properly, or does not generate figures, or the figures are not adequately formatted:

The plotting scripts of this POD may not produce the desired figures with the latest version of *matplotlib* (because of the default size adjustment settings). The *matplotlib* version comes with the Anaconda 2 installer, version 5.0.1 has been tested. The readers can switch to this older version. See section 2, 2).

Depending on the platform and Linux distribution/version, a related error may occur with the error message "…*ImportError: libcrypto.so.1.0.0: cannot open shared object file: No such file or directory*". One can find the missing object file *libcrypto.so.1.0.0* in the subdirectory

~/anaconda2/pkgs/openssl-1.0.2l-h077ae2c_5/lib/[8]

Manually copying the object file to ~/anaconda2/lib/ should solve the error.

[7] The folder name *ImageMagick* may depend on its version, e.g., *ImageMagick-6*.
[8] ~/anaconda2/ is where Anaconda 2 is installed. The precise names of the object file and openssl-folder may vary.