GCPy

Release 1.4.2

GEOS-Chem Support Team

Mar 14, 2024

BASIC USAGE OF GCPY:

Welcome to the GCPy ReadTheDocs documentation! This site provides documentation on the functionality of GCPy and instructions for common use cases.

GCPy is a Python-based toolkit containing useful functions for working specifically with the **GEOS-Chem** model of atmospheric chemistry and composition. GCPy is primarily used for plotting/tabling GEOS-Chem output and regridding input files in special cases.

For documentation on setting up and running GEOS-Chem please see our list of manuals for GEOS-Chem and related software.

CHAPTER

ONE

ABOUT GCPY

GCPy is a Python-based toolkit containing useful functions for working specifically with the **GEOS-Chem** model of atmospheric chemistry and composition.

GCPy aims to build on the well-established scientific Python technical stack, leveraging tools like **cartopy**, **numpy**, and **xarray** to simplify the task of working with GEOS-Chem model output and performing atmospheric chemistry analyses.

1.1 What GCPy was intended to do

1. Produce plots and tables from GEOS-Chem output using simple function calls.
2. Generate the standard evaluation plots and tables from GEOS-Chem benchmark simulations.
3. Obtain GEOS-Chem's horizontal and vertical grid information.
4. Implement GCHP-specific regridding functionalities (e.g. cubed-sphere to lat-lon regridding).
5. Provide example scripts for creating specific types of plots or analysis from GEOS-Chem output.
6. Provide user-submitted scripts for specific applications related to GEOS-Chem and HEMCO.

1.2 What GCPy was not intended to do

1. General NetCDF file modification: (crop a domain, extract some variables):
   - Instead, use netCDF tools such as:
     - xarray
     - netCDF Operators (NCO)
     - Climate Data Operators (CDO)
   - Also see our Work with netCDF data guide at geos-chem.readthedocs.io.
2. Statistical analysis:
   - Instead, use statistical tools such as:
     - scipy
     - scikit-learn
     - R
     - etc.

3. Machine Learning:
   - Instead, use machine learning tools such as:
     - pytorch
     - tensorflow
     - julia
     - etc.

1.3 License

GCPy is distributed under the MIT license. Please see the GCPy license agreement and List of GCPy developers for more information.

1.4 Requesting support

To report a bug or suggest a new feature, please see our Support Guidelines.

1.5 Submitting new features

If you are interested in submitting code to GCPy, please see our Contributing Guidelines.

CHAPTER TWO

INSTALLING GCPY

2.1 Requirements

GCPy is currently supported on the following platforms:

1. Linux (x86_64)
2. Windows Subsystem for Linux (running in Microsoft Windows 11)
3. MacOS

To install GCPy, you will need:

- **EITHER** a distribution of the **Mamba** package manager
- **OR** a distribution of the **Conda** package manager.

Mamba is a fast drop-in replacement for the widely-used **Conda** package manager. We recommend using **Mamba** to create a Python environment for GCPy. This environment will contain a version of the Python interpreter (in this case, Python 3.9) plus packages upon which GCPy depends.

Note: If your system has an existing **Conda** installation, and/or you do not wish to upgrade from **Conda** to **Mamba**, you may create the Python environment for GCPy with **Conda**. See the following sections for detailed instructions.

2.1.1 Check if Mamba is installed

Check if you already have **Mamba** on your system:

```
$ mamba --version
```

If **Mamba** has been installed, you will see output similar to this:

```
mamba version X.Y.Z
conda version A.B.C
```

If you see this output, you may skip ahead to the *Install GCPy and its dependencies* section.

2.1.2 Check if Conda is installed

If your system does not have **Mamba** installed, check if **Conda** is already present on your system:

```
$ conda --version
```

If a **Conda** version exists, you will see its version number printed to the screen:

```
conda version A.B.C
```

If neither **Conda** or **Mamba** are installed, we recommend installing the **Mamba** package manager yourself. Please proceed to the *Install MambaForge* section for instructions.

Additional setup for older Conda versions

If your **Conda** version is earlier than 23.7, you will need to do the following additional steps.

```
$ conda install -n base conda-libmamba-solver
$ conda config --set solver libmamba
```

This will install the fast **Mamba** environment solver into your **Conda** base environment. Using the **Mamba** solver within **Conda** will speed up the Python environment creation considerably.

Note: The **Mamba** environment solver is used by default in **Conda** 23.7 and later.

You may now skip ahead to the *Install GCPy and its dependencies* section.

2.2 Install MambaForge

We recommend installing the **MambaForge**, distribution, which is a full implementation of **Mamba** (as opposed to the minimal **MicroMamba** distribution).

Follow the instructions below to install **MambaForge**:

2.2.1 MacOS

1. Install **MambaForge** with Homebrew:

   ```
   $ brew install mambaforge
   ```

2. Initialize **Mamba** for your shell. Type one of the following commands:

   ```
   $ mamba init bash    # If you use the bash shell (recommended!)
   $ mamba init zsh     # If you use the zsh shell
   $ mamba init fish    # If you use the fish shell
   ```

   Mamba will add some code to your `~/.bash_profile` startup script that will tell your shell where to look for Python environments.

3. Exit your current terminal session and open a new terminal session. This will apply the changes.

You may now skip ahead to the *Install GCPy and its dependencies* section.

2.2.2 Linux and Windows Subsystem for Linux

1. Download the **MambaForge** installer script from the conda-forge GitHub releases page:

   ```
   $ wget https://github.com/conda-forge/miniforge/releases/download/23.3.1-0/
   ↪Mambaforge-23.3.1-0-Linux-x86_64.sh
   ```

   This will download the **MambaForge** installer script `Mambaforge-23.3.1-0-Linux-x86_64.sh` to your computer.

   Note: As of this writing (August 2023), the latest **MambaForge** version is `23.1.0-0`. If you find that the version has since been updated, simply replace the version number `23.3.1-0` in the above command with the most recent version number.

2. Change the permission of the **MambaForge** installer script so that it is executable.

   ```
   $ chmod 755 Mambaforge-23.3.1-0-Linux-x86_64.sh
   ```

3. Execute the **Mambaforge** installer script.

   ```
   $ ./Mambaforge-23.3.1-0-Linux-x86_64.sh
   ```

   To update an older version of **Mamba**, add the `-u` option to the above command.

4. Review and accept the license agreement.

   ```
   In order to continue the installation process, please review the license
   agreement.
   Please, press ENTER to continue
   >>>
   ```

   Press `ENTER` and then `SPACE` until you reach the end of the license agreement. Then you will be asked:

   ```
   Do you accept the license terms? [yes|no]
   [no] >>>
   ```

   Type `yes` and hit `ENTER`.

5. Specify the root installation path for **MambaForge**.

   ```
    Mambaforge will now be installed into this location:
   /home/YOUR-USER-NAME/mambaforge

   - Press ENTER to confirm the location
   - Press CTRL-C to abort the installation
   - Or specify a different location below
   [/home/YOUR-USER-NAME/mambaforge] >>>
   ```

In most cases, it should be OK to accept the default installation location. But on some systems, users may be encouraged to install software into a different location (e.g. if there is a faster filesystem available than the home directory filesystem). Consult your sysadmin or IT staff if you are unsure where to install **MambaForge**.

Press the `ENTER` key to accept the default installation path or type a new path and then press `ENTER`.

```
:program:`MambaForge` will downlad and install Python software
packages into the  :file:`pkgs` subfolder of the root
installation path.  Similarly, when you :ref:`create Python
environments <gcpy-install>`, these will be installed to the
:file:`envs` subfolder of the root installation path.
```

6. You may see this warning:

```
WARNING:
 You currently have a PYTHONPATH environment variable set. This may cause
 unexpected behavior when running the Python interpreter in Mambaforge.
 For best results, please verify that your PYTHONPATH only points to
 directories of packages that are compatible with the Python interpreter
 in Mambaforge: /home/YOUR-USER-NAMEb/mambaforge
```

As long as your `PYTHONPATH` environment variable only contains the path to the root-level GCPy folder, you may safely ignore this. (More on `PYTHONPATH` in the *next section*.)

7. Tell the installer to initialize **MambaForge**.

```
Do you wish the installer to initialize Mambaforge
by running conda init? [yes|no]
[no] >>>
```

Type `yes` and then `ENTER`. The installer script will add some code to your `~/.bashrc` system startup file that will tell your shell where to find Python environments.

8. Exit your current terminal session. Start a new terminal session to apply the updates. You are now ready to install GCPy.

2.3 Install GCPy and its dependencies

Once you have made sure that **Mamba** (or **Conda**) is present on your system, you may create a Python environment for GCPy. Follow these steps:

1. **Download the GCPy source code.**

   Create and go to the directory in which you would like to store GCPy. In this example we will store GCPy in your `$HOME/python/` path, but you can store it wherever you wish. You can also name the GCPy download whatever you want. In this example the GCPy directory is called `GCPy`.

```
$ cd $HOME/python
$ git clone https://github.com/geoschem/gcpy.git GCPy
$ cd GCPy
```

2. **Create a new Python virtual environment for GCPy.**

A Python virtual environment is a named set of Python installs, e.g. packages, that are independent of other virtual environments. Using an environment dedicated to GCPy is useful to maintain a set of package dependencies compatible with GCPy without interfering with Python packages you use for other work. You can create a Python virtual environment from anywhere on your system. It will be stored in your **Mamba** (or **Conda** installation rather than the directory from which you create it).

You can create a Python virtual environment using a file that lists all packages and their versions to be included in the environment. GCPy includes such as file, `environment.yml`, located in the top-level directory of the package.

Run one of the following commands at the command prompt to create a virtual environment for use with GCPy. You can name environment whatever you wish. This example names it `gcpy_env`.

```
$ mamba env create -n gcpy_env --file=environment.yml   # If using Mamba

$ conda env create -n gcpy_env --file=environment.yml   # If using Conda
```

A list of packages to be downloaded will be displayed. A confirmation message will ask you if you really wish to install all of the listed packages. Type **Y** to proceed or **n** to abort.

Once successfully created you can activate the environment with one of these commands:

```
$ mamba activate gcpy_env   # If using Mamba

$ conda activate gcpy_env   # If using Conda
```

To exit the environment, use one of these commands:

```
$ mamba deactivate   # If using Mamba

$ conda deactivate   # If using Conda
```

3. **Add GCPy to** `PYTHONPATH`

   The environment variable `PYTHONPATH` specifies the locations of Python libraries on your system that were not installed by **Mamba**.

   Add the path to your GCPy source code folder `~/.bashrc` file:

   ```
   export PYTHONPATH=$PYTHONPATH:$HOME/python/GCPy
   ```

   and then use

   ```
   $ source ~/.bashrc
   ```

   to apply the change.

4. **Set the** `MPLBACKEND` **environment variable**

   The environment variable `MPLBACKEND` specifies the X11 backend that the Matplotlib package will use to render plots to the screen.

   Add this line to your `~/.bashrc` file on your local PC/Mac and on any remote computer systems where you will use GCPy:

   ```
   export MPLBACKEND=tkagg
   ```

   And then use:

```
$ source ~/.bashrc
```

to apply the change.

5. **Perform a simple test:**

   Run the following commands in your terminal to check if the installation was succcesful.

```
$ source $HOME/.bashrc     # Alternatively close and reopen your terminal
$ echo $PYTHONPATH         # Check it contains path to your GCPy clone
$ mamba activate gcpy_env
$ mamba list               # Check it contains contents of gcpy env file
$ python
>>> import gcpy
```

If no error messages are displayed, you have successfully installed GCPy and its dependencies.

2.4 Upgrading GCPy versions

Sometimes the GCPy dependency list changes with a new GCPy version, either through the addition of new packages or a change in the minimum version. You can always update to the latest GCPy version from within you GCPy clone, and then update your virtual environment using the environment.yml file included in the package.

Run the following commands to update both your GCPy version to the latest available.

```
$ cd $HOME/python/GCPy
$ git fetch -p
$ git checkout main
$ git pull
```

You can also checkout an older version by doing the following:

```
$ cd $HOME/python/GCPy
$ git fetch -p
$ git tag
$ git checkout tags/version_you_want
```

Once you have the version you wish you use you can do the following commands to then update your virtual environment:

```
$ mamba activate gcpy_env
$ cd $HOME/python/GCPy
$ mamba env update --file environment.yml --prune
```

CHAPTER

THREE

OVERVIEW OF CAPABILITIES

This page outlines the capabilities of GCPy with links to detailed function documentation.

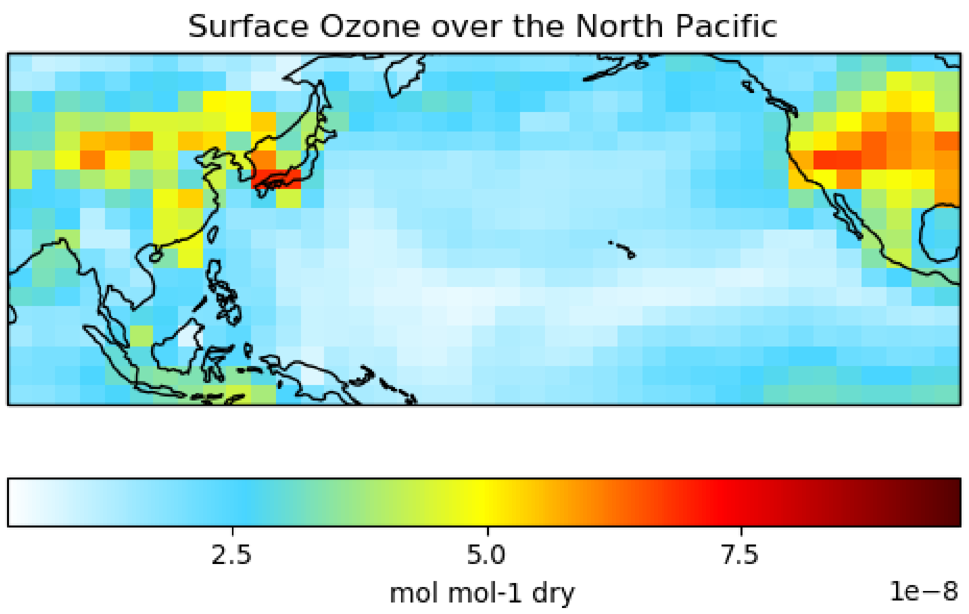
3.1 Spatial plotting

One hallmark of GCPy is easy-to-use spatial plotting of GEOS-Chem data. Available plotting falls into two layouts: single panel (one map of one variable from a dataset) and six panel (six maps comparing a variable between two datasets). The maps in these plots can display data at a single vertical level of your input dataset or in a zonal mean for all layers of the atmosphere.

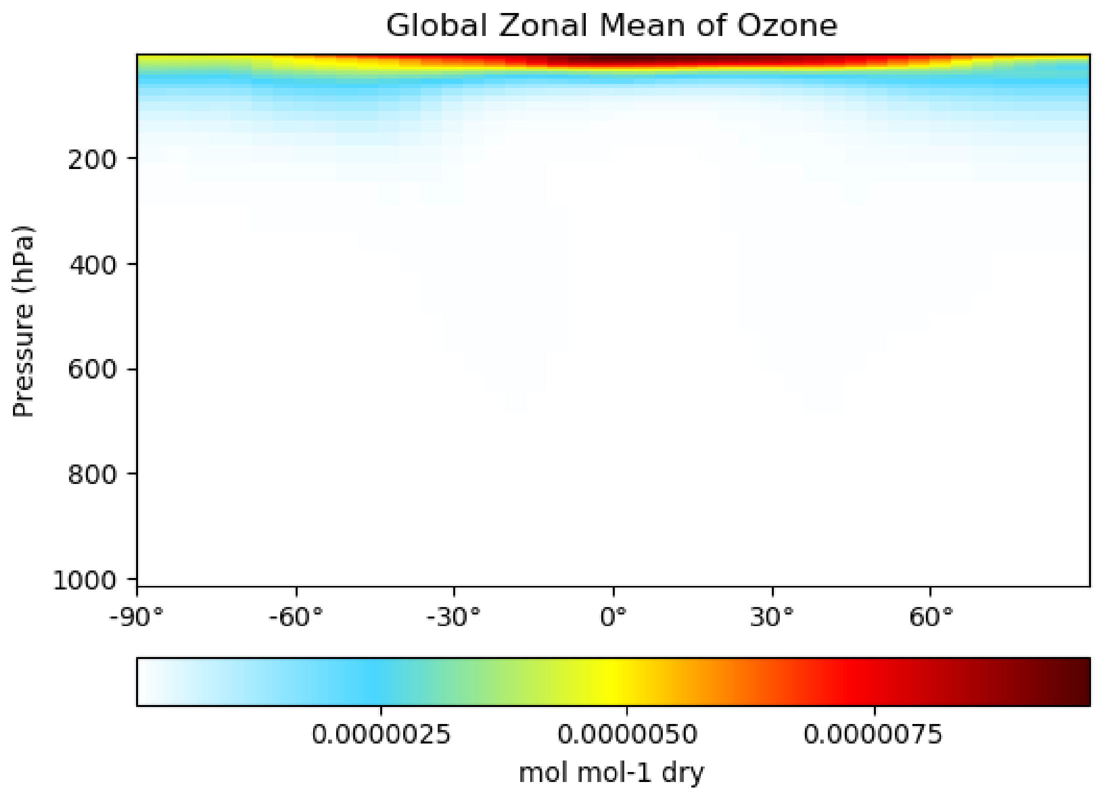
3.1.1 Single panel plots

Single panel plots are generated through the `single_panel()` function (located in module `gcpy.plot.single_panel`). This function uses Matplotlib and Cartopy plotting capabilities while handling certain behind the scenes operations that are necessary for plotting GEOS-Chem data, particularly for cubed-sphere and/or zonal mean data.

```
import xarray as xr
import matplotlib.pyplot as plt
from gcpy.plot.single_panel import single_panel

# Read data
ds = xr.open_dataset(
    'GEOSChem.Restart.20160701_0000z.nc4'
)

# Plot surface Ozone over the North Pacific
single_panel(
    ds['SpeciesRst_O3'].isel(lev=0),
    title='Surface Ozone over the North Pacific',
    extent=[80, -90, -10, 60]
)
plt.show()
```



```
# Plot global zonal mean of Ozone
single_panel(
    ds['SpeciesRst_O3'],
    plot_type='zonal_mean',
    title='Global Zonal Mean of Ozone'
)
plt.show()
```



Click here for an example single panel plotting script. *Click here* for detailed documentation for `single_panel()`.

3.1.2 Six-panel comparison plots

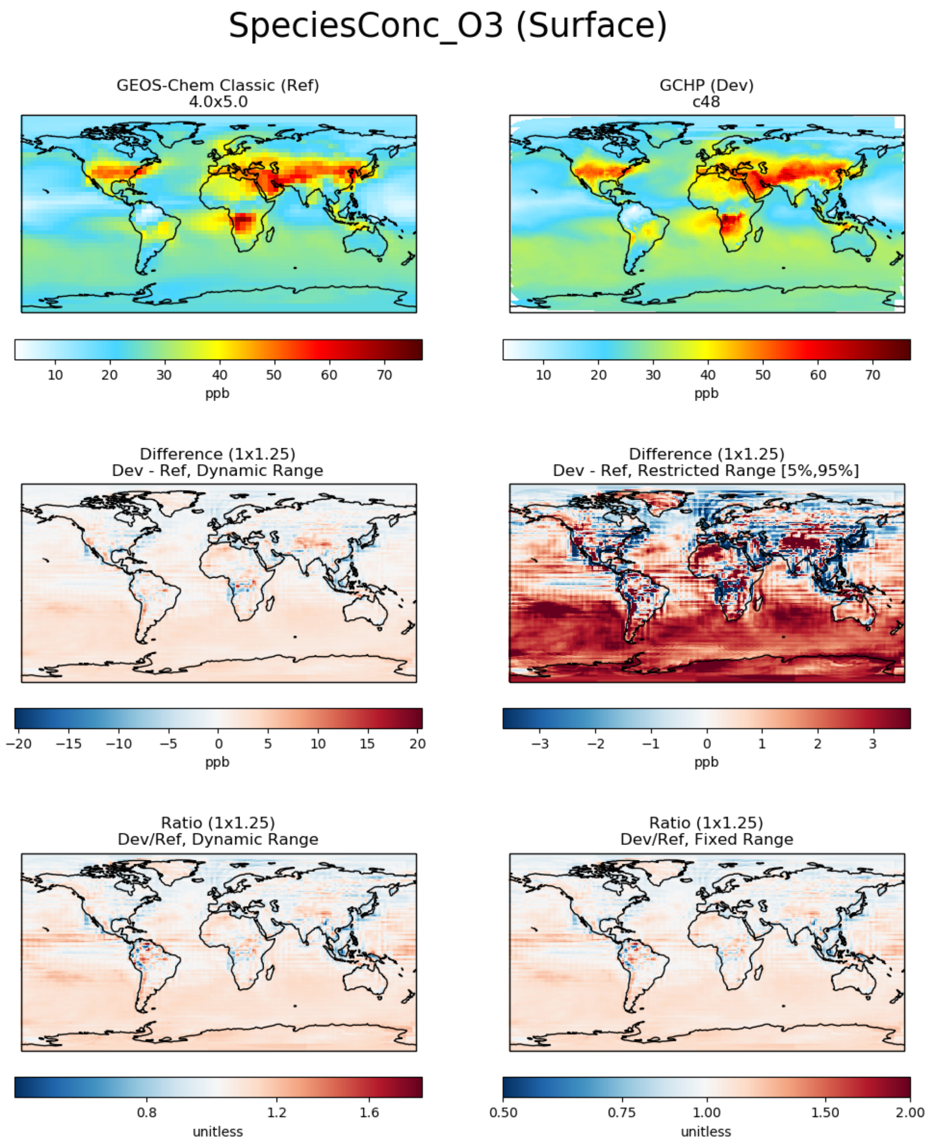
Six-panel plots are used to compare results across two different model runs. Single level and zonal mean plotting options are both available. The two model runs do not need to be the same resolution or even the same grid type (GEOS-Chem Classic and GCHP output can be mixed at will).

```
import xarray as xr
import matplotlib.pyplot as plt
from gcpy.plot.compare_single_level import compare_single_level
from gcpy.plot.compare_zonal_mean import compare_zonal_mean

# Read data
gcc_ds = xr.open_dataset(
    'GEOSChem.SpeciesConc.20160701_0000z.nc4'
)
gchp_ds = xr.open_dataset(
    'GCHP.SpeciesConc.20160716_1200z.nc4'
)

# Plot comparison of surface ozone over the North Pacific
compare_single_level(
```

(continues on next page)

(continued from previous page)

```
    gcc_ds,
    'GEOS-Chem Classic',
    gchp_ds,
    'GCHP',
    varlist=['SpeciesConc_O3'],
    extra_title_txt='Surface'
)
plt.show()
```

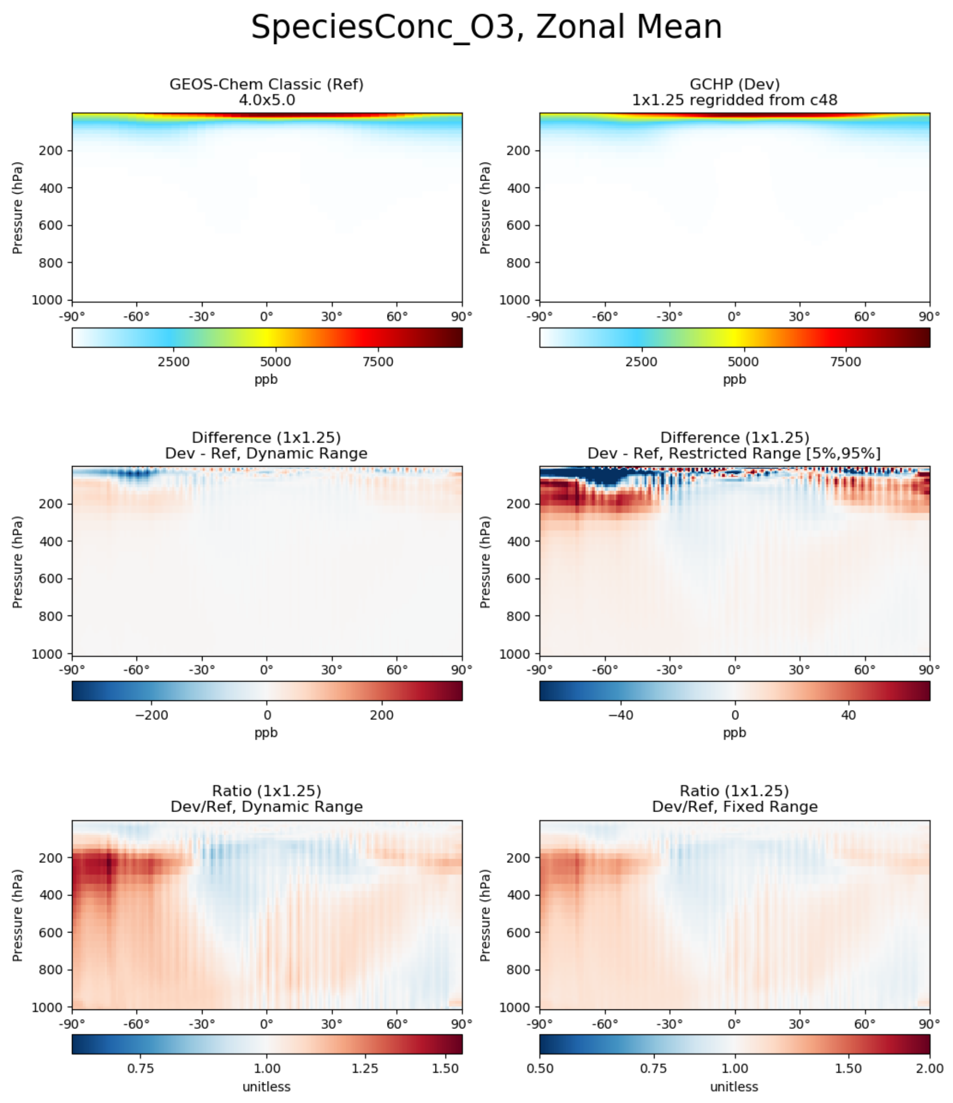


```
# Plot comparison of global zonal mean ozone
compare_zonal_mean(
    gcc_ds,
    'GEOS-Chem Classic',
```

(continues on next page)

(continued from previous page)

```
    gchp_ds,
    'GCHP',
    varlist=['SpeciesConc_O3']
)
plt.show()
```



Click here for an example six panel plotting script. *Click here* for complete documentation for `compare_single_level()` and `compare_zonal_mean()`.

3.1.3 Comprehensive benchmark plotting

The GEOS-Chem Support Team uses comprehensive plotting functions from module `gcpy.benchmark_funcs` to generate full plots of benchmark diagnostics. Functions like *gcpy.benchmark_funcs.make_benchmark_conc_plots()* by default create plots for every variable in a given collection (e.g. `SpeciesConc`) at multiple vertical levels (surface, 500hPa, zonal mean) and divide plots into separate folders based on category (e.g. Chlorine, Aerosols). The GEOS-Chem Support Team uses benchmark plotting and tabling table scripts (described in our *Benchmarking* chapter) to produce plots and tables for official model benchmarks.

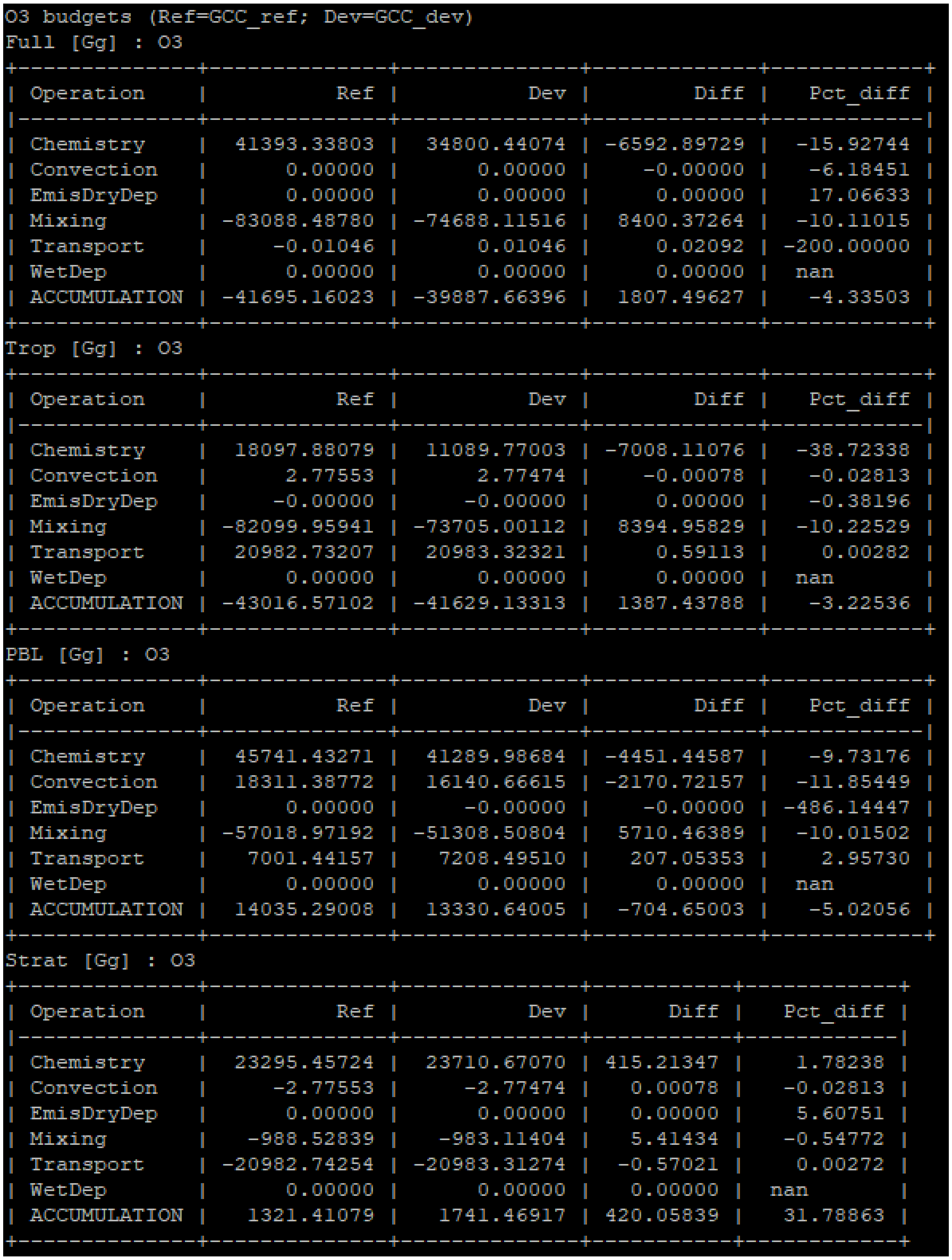
3.2 Table creation

GCPy has several dedicated functions for tabling GEOS-Chem output data in text file format. These functions and their outputs are primarily used for model benchmarking purposes.

3.2.1 Budget tables

Currently, budget tables can be created for "operations" (table shows change in mass after each category of model operation, as contained in the GEOS-Chem `Budget` diagnostics) or in overall averages for different aerosols or the Transport Tracers simulation.

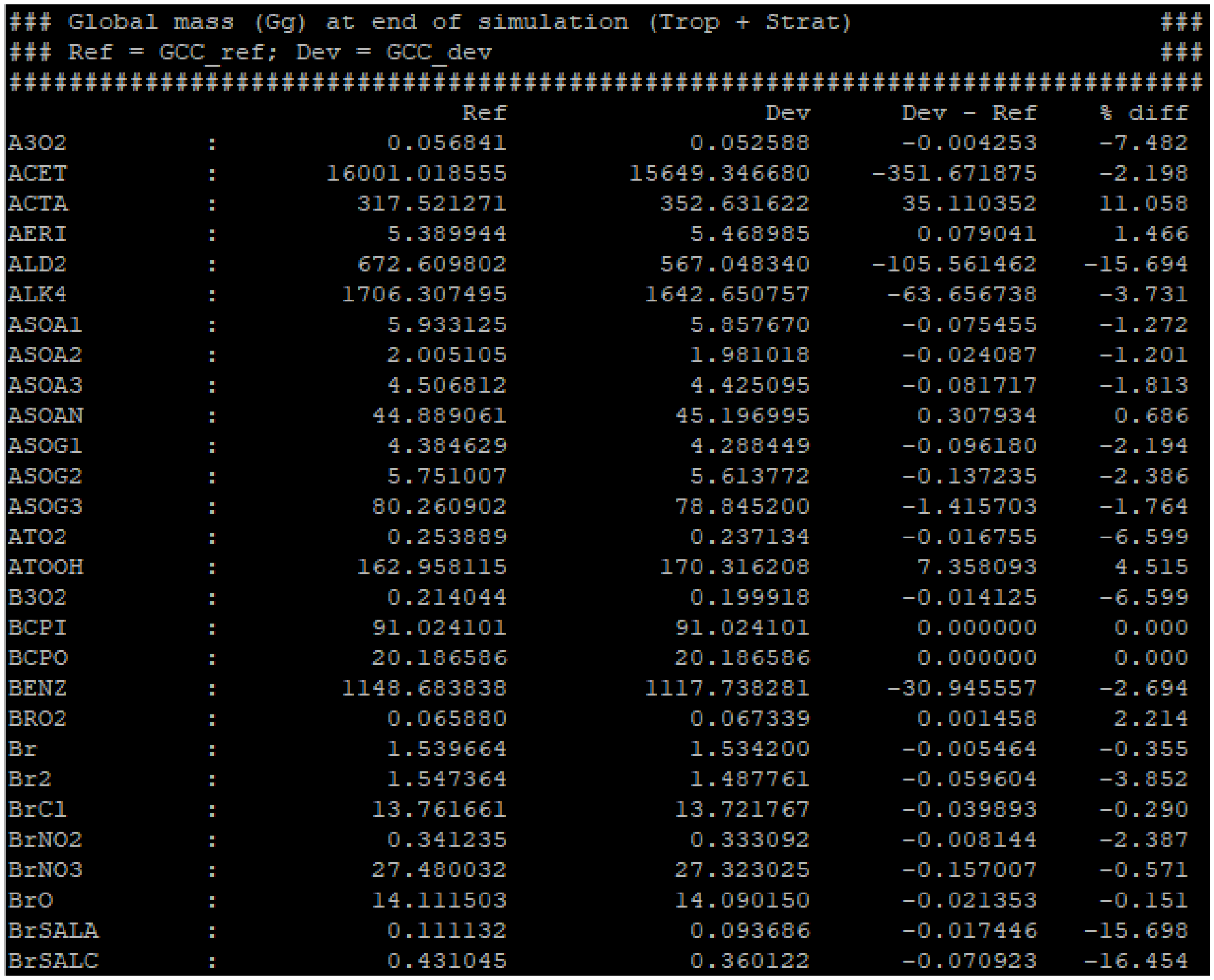
Operations budget tables are created using the *gcpy.benchmark_funcs.make_benchmark_operations_budget()* function and appear as follows:



```
O3 budgets (Ref=GCC_ref; Dev=GCC_dev)
Full [Gg] : O3
+--------------+--------------+--------------+-------------+------------+
| Operation    |          Ref |          Dev |        Diff |   Pct_diff |
|--------------+--------------+--------------+-------------+------------|
| Chemistry    |  41393.33803 |  34800.44074 | -6592.89729 |  -15.92744 |
| Convection   |      0.00000 |      0.00000 |    -0.00000 |   -6.18451 |
| EmisDryDep   |      0.00000 |      0.00000 |     0.00000 |   17.06633 |
| Mixing       | -83088.48780 | -74688.11516 |  8400.37264 |  -10.11015 |
| Transport    |     -0.01046 |      0.01046 |     0.02092 | -200.00000 |
| WetDep       |      0.00000 |      0.00000 |     0.00000 |  nan       |
| ACCUMULATION | -41695.16023 | -39887.66396 |  1807.49627 |   -4.33503 |
+--------------+--------------+--------------+-------------+------------+
Trop [Gg] : O3
+--------------+--------------+--------------+-------------+------------+
| Operation    |          Ref |          Dev |        Diff |   Pct_diff |
|--------------+--------------+--------------+-------------+------------|
| Chemistry    |  18097.88079 |  11089.77003 | -7008.11076 |  -38.72338 |
| Convection   |      2.77553 |      2.77474 |    -0.00078 |   -0.02813 |
| EmisDryDep   |     -0.00000 |     -0.00000 |     0.00000 |   -0.38196 |
| Mixing       | -82099.95941 | -73705.00112 |  8394.95829 |  -10.22529 |
| Transport    |  20982.73207 |  20983.32321 |     0.59113 |    0.00282 |
| WetDep       |      0.00000 |      0.00000 |     0.00000 |  nan       |
| ACCUMULATION | -43016.57102 | -41629.13313 |  1387.43788 |   -3.22536 |
+--------------+--------------+--------------+-------------+------------+
PBL [Gg] : O3
+--------------+--------------+--------------+-------------+------------+
| Operation    |          Ref |          Dev |        Diff |   Pct_diff |
|--------------+--------------+--------------+-------------+------------|
| Chemistry    |  45741.43271 |  41289.98684 | -4451.44587 |   -9.73176 |
| Convection   |  18311.38772 |  16140.66615 | -2170.72157 |  -11.85449 |
| EmisDryDep   |      0.00000 |     -0.00000 |    -0.00000 | -486.14447 |
| Mixing       | -57018.97192 | -51308.50804 |  5710.46389 |  -10.01502 |
| Transport    |   7001.44157 |   7208.49510 |   207.05353 |    2.95730 |
| WetDep       |      0.00000 |      0.00000 |     0.00000 |  nan       |
| ACCUMULATION |  14035.29008 |  13330.64005 |  -704.65003 |   -5.02056 |
+--------------+--------------+--------------+-------------+------------+
Strat [Gg] : O3
+--------------+--------------+--------------+-----------+------------+
| Operation    |          Ref |          Dev |      Diff |   Pct_diff |
|--------------+--------------+--------------+-----------+------------|
| Chemistry    |  23295.45724 |  23710.67070 | 415.21347 |    1.78238 |
| Convection   |     -2.77553 |     -2.77474 |   0.00078 |   -0.02813 |
| EmisDryDep   |      0.00000 |      0.00000 |   0.00000 |    5.60751 |
| Mixing       |   -988.52839 |   -983.11404 |   5.41434 |   -0.54772 |
| Transport    | -20982.74254 | -20983.31274 |  -0.57021 |    0.00272 |
| WetDep       |      0.00000 |      0.00000 |   0.00000 |  nan       |
| ACCUMULATION |   1321.41079 |   1741.46917 | 420.05839 |   31.78863 |
+--------------+--------------+--------------+-----------+------------+
```

3.2.2 Mass tables

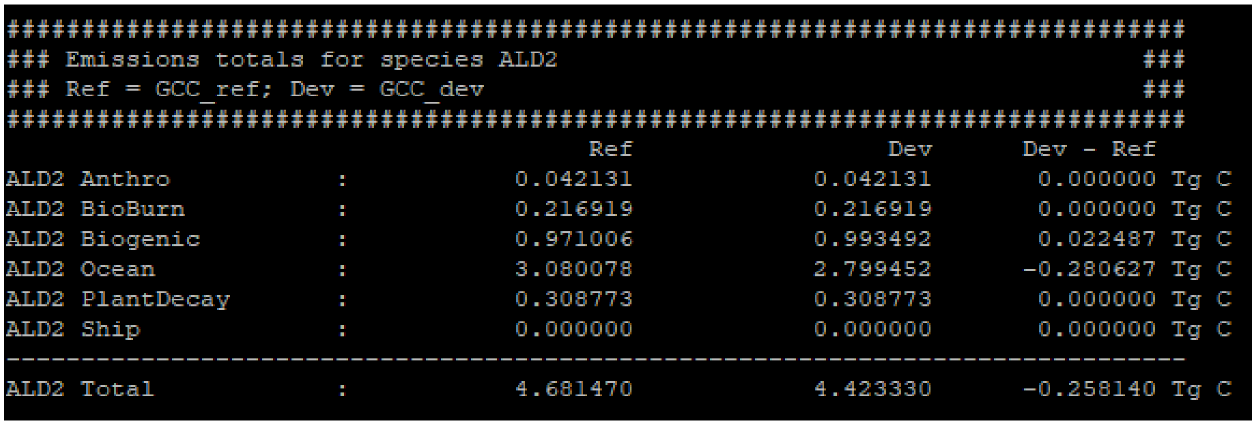
The *gcpy.benchmark_funcs.make_benchmark_mass_tables()* function uses species concentrations and info from meteorology files to generate the total mass of species in certain segments of the atmosphere (currently global or only the troposphere). An example table is shown below:



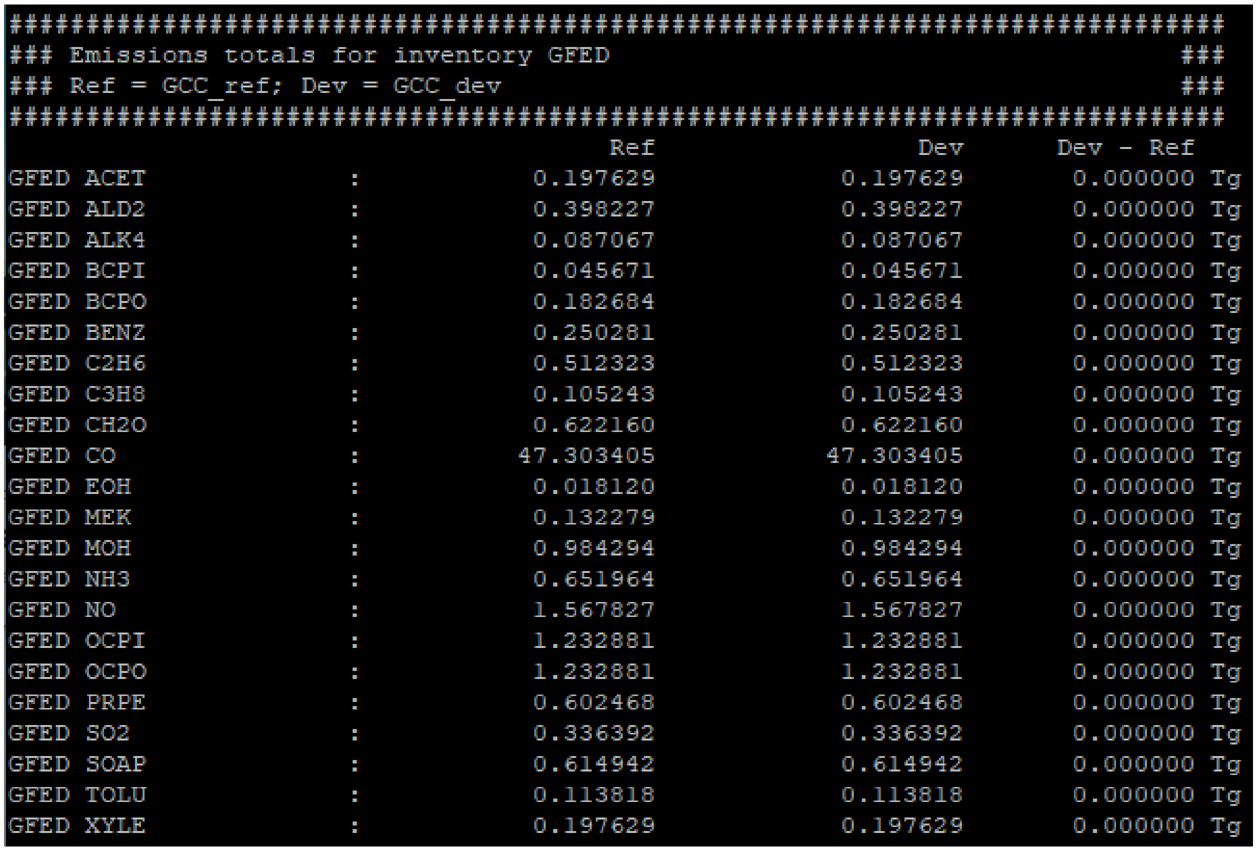
```
### Global mass (Gg) at end of simulation (Trop + Strat)                      ###
### Ref = GCC_ref; Dev = GCC_dev                                              ###
################################################################################
                              Ref                 Dev       Dev - Ref    % diff
A3O2         :           0.056841            0.052588       -0.004253    -7.482
ACET         :       16001.018555        15649.346680     -351.671875    -2.198
ACTA         :         317.521271          352.631622       35.110352    11.058
AERI         :           5.389944            5.468985        0.079041     1.466
ALD2         :         672.609802          567.048340     -105.561462   -15.694
ALK4         :        1706.307495         1642.650757      -63.656738    -3.731
ASOA1        :           5.933125            5.857670       -0.075455    -1.272
ASOA2        :           2.005105            1.981018       -0.024087    -1.201
ASOA3        :           4.506812            4.425095       -0.081717    -1.813
ASOAN        :          44.889061           45.196995        0.307934     0.686
ASOG1        :           4.384629            4.288449       -0.096180    -2.194
ASOG2        :           5.751007            5.613772       -0.137235    -2.386
ASOG3        :          80.260902           78.845200       -1.415703    -1.764
ATO2         :           0.253889            0.237134       -0.016755    -6.599
ATOOH        :         162.958115          170.316208        7.358093     4.515
B3O2         :           0.214044            0.199918       -0.014125    -6.599
BCPI         :          91.024101           91.024101        0.000000     0.000
BCPO         :          20.186586           20.186586        0.000000     0.000
BENZ         :        1148.683838         1117.738281      -30.945557    -2.694
BRO2         :           0.065880            0.067339        0.001458     2.214
Br           :           1.539664            1.534200       -0.005464    -0.355
Br2          :           1.547364            1.487761       -0.059604    -3.852
BrCl         :          13.761661           13.721767       -0.039893    -0.290
BrNO2        :           0.341235            0.333092       -0.008144    -2.387
BrNO3        :          27.480032           27.323025       -0.157007    -0.571
BrO          :          14.111503           14.090150       -0.021353    -0.151
BrSALA       :           0.111132            0.093686       -0.017446   -15.698
BrSALC       :           0.431045            0.360122       -0.070923   -16.454
```

3.2.3 Emissions tables

The *gcpy.benchmark_funcs.make_benchmark_emis_tables()* function creates tables of total emissions categorized by species or by inventory. Examples of both emissions table types are shown below:



```
################################################################################
### Emissions totals for species ALD2                                        ###
### Ref = GCC_ref; Dev = GCC_dev                                             ###
################################################################################
                                             Ref                 Dev      Dev - Ref
ALD2 Anthro           :            0.042131            0.042131        0.000000 Tg C
ALD2 BioBurn          :            0.216919            0.216919        0.000000 Tg C
ALD2 Biogenic         :            0.971006            0.993492        0.022487 Tg C
ALD2 Ocean            :            3.080078            2.799452       -0.280627 Tg C
ALD2 PlantDecay       :            0.308773            0.308773        0.000000 Tg C
ALD2 Ship             :            0.000000            0.000000        0.000000 Tg C
--------------------------------------------------------------------------------
ALD2 Total            :            4.681470            4.423330       -0.258140 Tg C
```



```
################################################################################
### Emissions totals for inventory GFED                                      ###
### Ref = GCC_ref; Dev = GCC_dev                                             ###
################################################################################
                                             Ref                 Dev      Dev - Ref
GFED ACET             :            0.197629            0.197629        0.000000 Tg
GFED ALD2             :            0.398227            0.398227        0.000000 Tg
GFED ALK4             :            0.087067            0.087067        0.000000 Tg
GFED BCPI             :            0.045671            0.045671        0.000000 Tg
GFED BCPO             :            0.182684            0.182684        0.000000 Tg
GFED BENZ             :            0.250281            0.250281        0.000000 Tg
GFED C2H6             :            0.512323            0.512323        0.000000 Tg
GFED C3H8             :            0.105243            0.105243        0.000000 Tg
GFED CH2O             :            0.622160            0.622160        0.000000 Tg
GFED CO               :           47.303405           47.303405        0.000000 Tg
GFED EOH              :            0.018120            0.018120        0.000000 Tg
GFED MEK              :            0.132279            0.132279        0.000000 Tg
GFED MOH              :            0.984294            0.984294        0.000000 Tg
GFED NH3              :            0.651964            0.651964        0.000000 Tg
GFED NO               :            1.567827            1.567827        0.000000 Tg
GFED OCPI             :            1.232881            1.232881        0.000000 Tg
GFED OCPO             :            1.232881            1.232881        0.000000 Tg
GFED PRPE             :            0.602468            0.602468        0.000000 Tg
GFED SO2              :            0.336392            0.336392        0.000000 Tg
GFED SOAP             :            0.614942            0.614942        0.000000 Tg
GFED TOLU             :            0.113818            0.113818        0.000000 Tg
GFED XYLE             :            0.197629            0.197629        0.000000 Tg
```

3.3 Regridding

3.3.1 General regridding rules

GCPy supports regridding between all horizontal GEOS-Chem grid types, including latitude/longitude grids (the grid format of GEOS-Chem Classic), standard cubed-sphere (the standard grid format of GCHP), and stretched-grid (an optional grid format in GCHP). GCPy contains several horizontal regridding functions built off of xESMF. GCPy automatically handles most regridding needs when plotting GEOS-Chem data.

gcpy.file_regrid() allows you to regrid GEOS-Chem Classic and GCHP files between different grid resolutions and can be called from the command line or as a function.

gcpy.regrid_restart_file allows you to regrid GCHP files between between different grid resolutions and grid types (standard and stretched cubed-sphere grids), and can be called from the command line.

The 72-level and 47-level vertical grids are pre-defined in GCPy. Other vertical grids can also be defined if you provide the A and B coefficients of the hybrid vertical grid.

When plotting data of differing grid types or horizontal resolutions using compare_single_level() or compare_zonal_mean(), you can specify a comparison resolution using the `cmpres` argument. This resolution will be used for the difference panels in each plot (the bottom four panels rather than the top two raw data panels). If you do not specify a comparison resolution, GCPy will automatically choose one.

For more extensive regridding information, visit the *detailed regridding documentation*.

CHAPTER

FOUR

PLOTTING

This page describes in depth the general plotting capabilities of GCPy, including possible argument values for every plotting function.

For information about GCPy functions that are specific to the GEOS-Chem benchmark workflow, please see our *Benchmarking* chapter.

4.1 Six-panel comparison plots

The functions listed below generate six-panel plots comparing variables between two datasets:

Plotting function	Located in GCPy module
`compare_single_level()`	`gcpy.plot.compare_single_level`
`compare_zonal_mean()`	`gcpy.plot.compare_zonal_mean`

Both `compare_single_level()` and `compare_zonal_mean()` generate a six panel plot for each variable passed. These plots can either be saved to PDFs or generated sequentially for visualization in the Matplotlib GUI using `matplotlib.pyplot.show()`. Each plot uses data passed from a reference (`Ref`) dataset and a development (`Dev`) dataset. Both functions share significant structural overlap both in output appearance and code implementation.

You can import these routines into your code with these statements:

```
from gcpy.plot.compare_single_level import compare_single_level
from gcpy.plot.compare_zonal_mean import compare_zonal_mean
```

Each panel has a title describing the type of panel, a colorbar for the values plotted in that panel, and the units of the data plotted in that panel. The upper two panels of each plot show actual values from the `Ref` (left) and `Dev` (right) datasets for a given variable. The middle two panels show the difference (`Dev - Ref`) between the values in the `Dev` dataset and the values in the `Ref` dataset. The left middle panel uses a full dynamic color map, while the right middle panel caps the color map at the 5th and 95th percentiles. The bottom two panels show the ratio (`Dev/Ref`) between the values in the Dev dataset and the values in the Ref Dataset. The left bottom panel uses a full dynamic color map, while the right bottom panel caps the color map at 0.5 and 2.0.

4.1.1 Function `compare_single_level`

The `compare_single_level` function accepts takes the following arguments:

```
def compare_single_level(
        refdata,
        refstr,
        devdata,
        devstr,
        varlist=None,
        ilev=0,
        itime=0,
        refmet=None,
        devmet=None,
        weightsdir='.',
        pdfname="",
        cmpres=None,
        match_cbar=True,
        normalize_by_area=False,
        enforce_units=True,
        convert_to_ugm3=False,
        flip_ref=False,
        flip_dev=False,
        use_cmap_RdBu=False,
        verbose=False,
        log_color_scale=False,
        extra_title_txt=None,
        extent=None,
        n_job=-1,
        sigdiff_list=None,
        second_ref=None,
        second_dev=None,
        spcdb_dir=os.path.dirname(__file__),
        sg_ref_path='',
        sg_dev_path='',
        ll_plot_func='imshow',
        **extra_plot_args
):
    """
    Create single-level 3x2 comparison map plots for variables common
    in two xarray Datasets. Optionally save to PDF.

    Args:
        refdata: xarray dataset
            Dataset used as reference in comparison
        refstr: str
            String description for reference data to be used in plots
        devdata: xarray dataset
            Dataset used as development in comparison
        devstr: str
            String description for development data to be used in plots

    Keyword Args (optional):
```

(continues on next page)

(continued from previous page)

```
        varlist: list of strings
            List of xarray dataset variable names to make plots for
            Default value: None (will compare all common variables)
        ilev: integer
            Dataset level dimension index using 0-based system.
            Indexing is ambiguous when plotting differing vertical grids
            Default value: 0
        itime: integer
            Dataset time dimension index using 0-based system
            Default value: 0
        refmet: xarray dataset
            Dataset containing ref meteorology
            Default value: None
        devmet: xarray dataset
            Dataset containing dev meteorology
            Default value: None
        weightsdir: str
            Directory path for storing regridding weights
            Default value: None (will create/store weights in
            current directory)
        pdfname: str
            File path to save plots as PDF
            Default value: Empty string (will not create PDF)
        cmpres: str
            String description of grid resolution at which
            to compare datasets
            Default value: None (will compare at highest resolution
            of ref and dev)
        match_cbar: bool
            Set this flag to True if you wish to use the same colorbar
            bounds for the Ref and Dev plots.
            Default value: True
        normalize_by_area: bool
            Set this flag to True if you wish to normalize the Ref
            and Dev raw data by grid area. Input ref and dev datasets
            must include AREA variable in m2 if normalizing by area.
            Default value: False
        enforce_units: bool
            Set this flag to True to force an error if Ref and Dev
            variables have different units.
            Default value: True
        convert_to_ugm3: bool
            Whether to convert data units to ug/m3 for plotting.
            Default value: False
        flip_ref: bool
            Set this flag to True to flip the vertical dimension of
            3D variables in the Ref dataset.
            Default value: False
        flip_dev: bool
            Set this flag to True to flip the vertical dimension of
            3D variables in the Dev dataset.
            Default value: False
```

(continues on next page)

(continued from previous page)

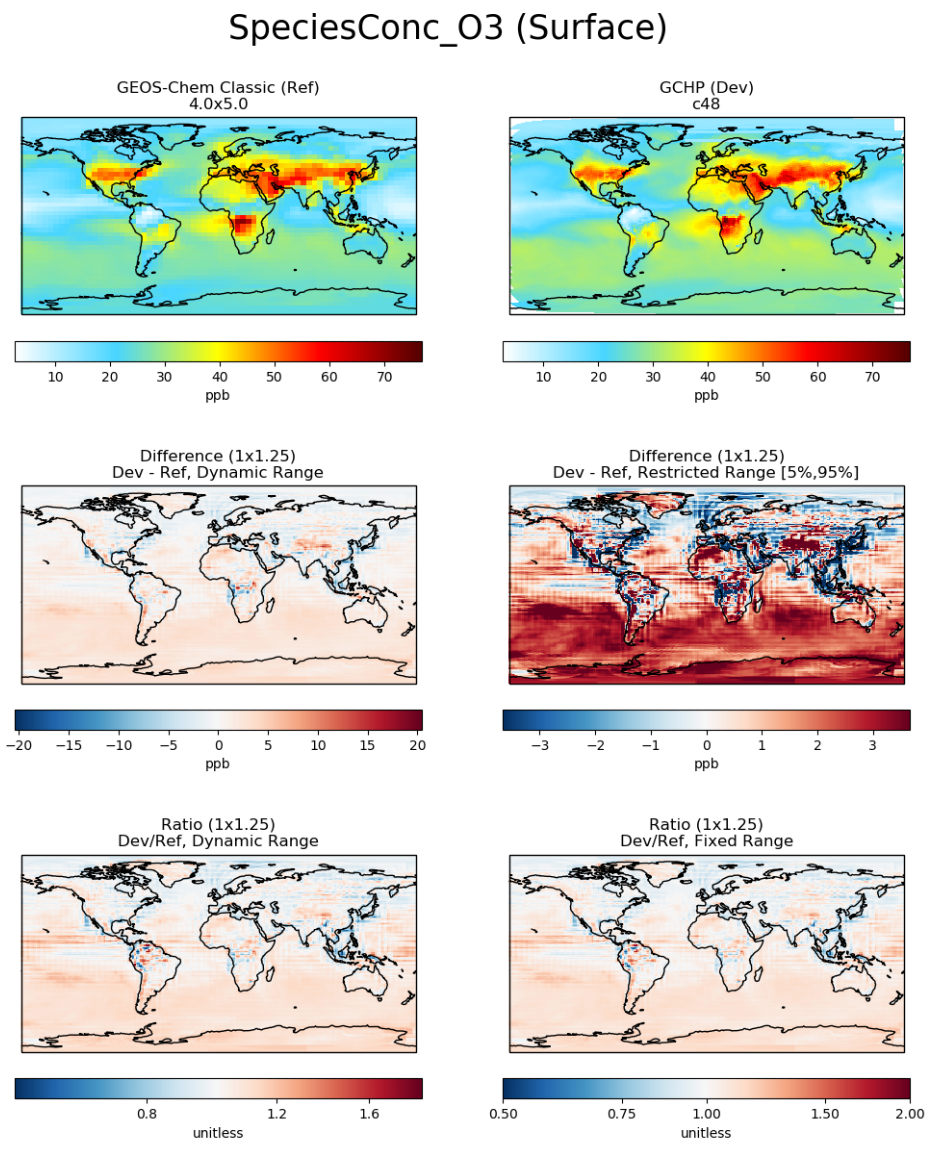
```
        use_cmap_RdBu: bool
            Set this flag to True to use a blue-white-red colormap
            for plotting the raw data in both the Ref and Dev datasets.
            Default value: False
        verbose: bool
            Set this flag to True to enable informative printout.
            Default value: False
        log_color_scale: bool
            Set this flag to True to plot data (not diffs)
            on a log color scale.
            Default value: False
        extra_title_txt: str
            Specifies extra text (e.g. a date string such as "Jan2016")
            for the top-of-plot title.
            Default value: None
        extent: list
            Defines the extent of the region to be plotted in form
            [minlon, maxlon, minlat, maxlat].
            Default value plots extent of input grids.
            Default value: [-1000, -1000, -1000, -1000]
        n_job: int
            Defines the number of simultaneous workers for parallel
            plotting.  Set to 1 to disable parallel plotting.
            Value of -1 allows the application to decide.
            Default value: -1
        sigdiff_list: list of str
            Returns a list of all quantities having significant
            differences (where |max(fractional difference)| > 0.1).
            Default value: None
        second_ref: xarray Dataset
            A dataset of the same model type / grid as refdata,
            to be used in diff-of-diffs plotting.
            Default value: None
        second_dev: xarray Dataset
            A dataset of the same model type / grid as devdata,
            to be used in diff-of-diffs plotting.
            Default value: None
        spcdb_dir: str
            Directory containing species_database.yml file.
            Default value: Path of GCPy code repository
        sg_ref_path: str
            Path to NetCDF file containing stretched-grid info
            (in attributes) for the ref dataset
            Default value: '' (will not be read in)
        sg_dev_path: str
            Path to NetCDF file containing stretched-grid info
            (in attributes) for the dev dataset
            Default value: '' (will not be read in)
        ll_plot_func: str
            Function to use for lat/lon single level plotting with
            possible values 'imshow' and 'pcolormesh'. imshow is much
            faster but is slightly displaced when plotting from
```

(continues on next page)

(continued from previous page)

```
                dateline to dateline and/or pole to pole.
                Default value: 'imshow'
            extra_plot_args: various
                Any extra keyword arguments are passed through the
                plotting functions to be used in calls to pcolormesh() (CS)
                or imshow() (Lat/Lon).
"""
```

and generates a comparison plot such as:



4.1.2 Function `compare_zonal_mean`

```
def compare_zonal_mean(
        refdata,
        refstr,
        devdata,
        devstr,
        varlist=None,
        itime=0,
        refmet=None,
        devmet=None,
        weightsdir='.',
        pdfname="",
        cmpres=None,
        match_cbar=True,
        pres_range=None,
        normalize_by_area=False,
        enforce_units=True,
        convert_to_ugm3=False,
        flip_ref=False,
        flip_dev=False,
        use_cmap_RdBu=False,
        verbose=False,
        log_color_scale=False,
        log_yaxis=False,
        extra_title_txt=None,
        n_job=-1,
        sigdiff_list=None,
        second_ref=None,
        second_dev=None,
        spcdb_dir=os.path.dirname(__file__),
        sg_ref_path='',
        sg_dev_path='',
        ref_vert_params=None,
        dev_vert_params=None,
        **extra_plot_args
):
    """
    Creates 3x2 comparison zonal-mean plots for variables
    common in two xarray Datasets. Optionally save to PDF.

    Args:
        refdata: xarray dataset
            Dataset used as reference in comparison
        refstr: str
            String description for reference data to be used in plots
        devdata: xarray dataset
            Dataset used as development in comparison
        devstr: str
            String description for development data to be used in plots

    Keyword Args (optional):
        varlist: list of strings
```

(continues on next page)

(continued from previous page)

```
        List of xarray dataset variable names to make plots for
        Default value: None (will compare all common 3D variables)
    itime: integer
        Dataset time dimension index using 0-based system
        Default value: 0
    refmet: xarray dataset
        Dataset containing ref meteorology
        Default value: None
    devmet: xarray dataset
        Dataset containing dev meteorology
        Default value: None
    weightsdir: str
        Directory path for storing regridding weights
        Default value: None (will create/store weights in
        current directory)
    pdfname: str
        File path to save plots as PDF
        Default value: Empty string (will not create PDF)
    cmpres: str
        String description of grid resolution at which
        to compare datasets
        Default value: None (will compare at highest resolution
        of Ref and Dev)
    match_cbar: bool
        Set this flag to True to use same the colorbar bounds
        for both Ref and Dev plots.
        Default value: True
    pres_range: list of two integers
        Pressure range of levels to plot [hPa]. The vertical axis
        will span the outer pressure edges of levels that contain
        pres_range endpoints.
        Default value: [0, 2000]
    normalize_by_area: bool
        Set this flag to True to to normalize raw data in both
        Ref and Dev datasets by grid area. Input ref and dev
        datasets must include AREA variable in m2 if normalizing
        by area.
        Default value: False
    enforce_units: bool
        Set this flag to True force an error if the variables in
        the Ref and Dev datasets have different units.
        Default value: True
    convert_to_ugm3: str
        Whether to convert data units to ug/m3 for plotting.
        Default value: False
    flip_ref: bool
        Set this flag to True to flip the vertical dimension of
        3D variables in the Ref dataset.
        Default value: False
    flip_dev: bool
        Set this flag to True to flip the vertical dimension of
        3D variables in the Dev dataset.
```

(continues on next page)

(continued from previous page)

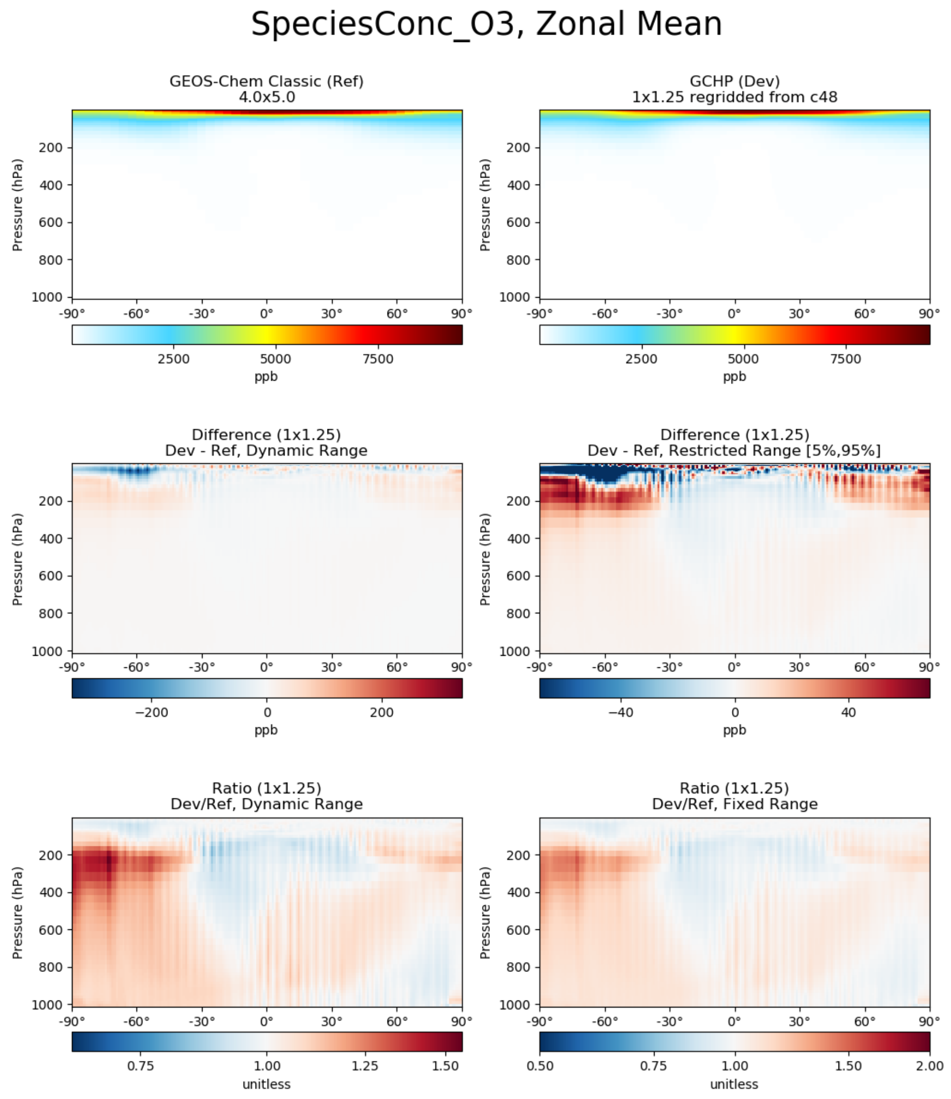
```
        Default value: False
    use_cmap_RdBu: bool
        Set this flag to True to use a blue-white-red colormap for
        plotting raw reference and development datasets.
        Default value: False
    verbose: logical
        Set this flag to True to enable informative printout.
        Default value: False
    log_color_scale: bool
        Set this flag to True to enable plotting data (not diffs)
        on a log color scale.
        Default value: False
    log_yaxis: bool
        Set this flag to True if you wish to create zonal mean
        plots with a log-pressure Y-axis.
        Default value: False
    extra_title_txt: str
        Specifies extra text (e.g. a date string such as "Jan2016")
        for the top-of-plot title.
        Default value: None
    n_job: int
        Defines the number of simultaneous workers for parallel
        plotting.  Set to 1 to disable parallel plotting.
        Value of -1 allows the application to decide.
        Default value: -1
    sigdiff_list: list of str
        Returns a list of all quantities having significant
        differences (where |max(fractional difference)| > 0.1).
        Default value: None
    second_ref: xarray Dataset
        A dataset of the same model type / grid as refdata,
        to be used in diff-of-diffs plotting.
        Default value: None
    second_dev: xarray Dataset
        A dataset of the same model type / grid as devdata,
        to be used in diff-of-diffs plotting.
        Default value: None
    spcdb_dir: str
        Directory containing species_database.yml file.
        Default value: Path of GCPy code repository
    sg_ref_path: str
        Path to NetCDF file containing stretched-grid info
        (in attributes) for the ref dataset
        Default value: '' (will not be read in)
    sg_dev_path: str
        Path to NetCDF file containing stretched-grid info
        (in attributes) for the dev dataset
        Default value: '' (will not be read in)
    ref_vert_params: list(AP, BP) of list-like types
        Hybrid grid parameter A in hPa and B (unitless).
        Needed if ref grid is not 47 or 72 levels.
        Default value: None
```

(continues on next page)

(continued from previous page)

```
        dev_vert_params: list(AP, BP) of list-like types
            Hybrid grid parameter A in hPa and B (unitless).
            Needed if dev grid is not 47 or 72 levels.
            Default value: None
        extra_plot_args: various
            Any extra keyword arguments are passed through the
            plotting functions to be used in calls to pcolormesh()
            (CS) or imshow() (Lat/Lon).
    """
```

and generates a comparison plot such as:



4.1.3 Shared structure

Both `compare_single_level()` and `compare_zonal_mean()` have four positional (required) arguments.

refdata : `xarray.Dataset`
: Dataset used as reference in comparison

refstr : `str OR list of str`
: String description for reference data to be used in plots OR list containing [ref1str, ref2str] for diff-of-diffs plots

devdata : `xarray.Dataset`
: Dataset used as development in comparison

devstr : `str OR list of str`
: String description for development data to be used in plots OR list containing [dev1str, dev2str] for diff-of-diffs plots

refstr and *devstr* title the top two panels of each six panel plot.

Functions `compare_single_level()` and `compare_zonal_mean()` share many arguments. Some of these arguments are plotting options that change the format of the plots:

For example, you may wish to convert units to ug/m^3 when generating comparison plots of aerosol species. Activate this option by setting the keyword argument `convert_to_ugm3=True`.

Other arguments are necessary to achieve a correct plot depending on the format of `refdata` and `devdata` and require you to know certain traits of your input data. For example, you must specify if one of the datasets should be flipped vertically if Z coordinates in that dataset do not denote decreasing pressure as Z index increases, otherwise the vertical coordinates between your two datasets may be misaligned and result in an undesired plotting outcome. This may be done with by setting the boolean options `flip_ref=True` and/or `flip_dev=True`.

The `n_job` argument governs the parallel plotting settings of `compare_single_level()` and `compare_zonal_mean()` . GCPy uses the JobLib library to create plots in parallel. Due to limitations with matplotlib, this parallelization creates plots (pages) in parallel rather than individual panels on a single page. Parallel plot creation is not enabled when you do not save to a PDF. The default value of `n_job=-1` allows the function call to automatically scale up to, at most, the number of cores available on your system.

Note: On systems with higher (12+) core counts, the maximum number of cores is not typically reached because of the process handling mechanics of JobLib. However, on lower-end systems with lower core counts or less available memory, it is advantageous to use `n_job` to limit the max number of processes.

Due to how Python handles memory management on Linux systems, using more cores may result in memory not returned to the system after the plots are created. Requesting fewer cores with `n_job` may help to avoid this situation.

4.1.4 Example script

Here is a basic script that calls both `compare_zonal_mean()` and `compare_single_level()`:

```python
#!/usr/bin/env python

import xarray as xr
import matplotlib.pyplot as plt
from gcpy.plot.compare_single_level import compare_single_level
from gcpy.plot.compare_zonal_mean import compare_zonal_mean

```

(continues on next page)

(continued from previous page)

```
file1 = '/path/to/ref'
file2 = '/path/to/dev'
ds1 = xr.open_dataset(file1)
ds2 = xr.open_dataset(file2)
compare_zonal_mean(ds1, 'Ref run', ds2, 'Dev run')
plt.show()
compare_single_level(ds1, 'Ref run', ds2, 'Dev run')
plt.show()
```

4.2 Single panel plots

Function `single_panel()` (contained in GCPy module `gcpy.plot.single_panel`) is used to create plots containing only one panel of GEOS-Chem data. This function is used within `compare_single_level()` and `compare_zonal_mean()` to generate each panel plot. It can also be called directly on its own to quickly plot GEOS-Chem data in zonal mean or single level format.

4.2.1 Function: `single_panel`

Function `single_panel()` accepts the following arguments:

```
def single_panel(
        plot_vals,
        ax=None,
        plot_type="single_level",
        grid=None,
        gridtype="",
        title="fill",
        comap=WhGrYlRd,
        norm=None,
        unit="",
        extent=None,
        masked_data=None,
        use_cmap_RdBu=False,
        log_color_scale=False,
        add_cb=True,
        pres_range=None,
        pedge=np.full((1, 1), -1),
        pedge_ind=np.full((1, 1), -1),
        log_yaxis=False,
        xtick_positions=None,
        xticklabels=None,
        proj=ccrs.PlateCarree(),
        sg_path='',
        ll_plot_func="imshow",
        vert_params=None,
        pdfname="",
        weightsdir='.',
        vmin=None,
        vmax=None,
```

(continues on next page)

(continued from previous page)

```
        return_list_of_plots=False,
        **extra_plot_args
):
    """
    Core plotting routine -- creates a single plot panel.

    Args:
        plot_vals: xarray.DataArray, numpy.ndarray, or dask.array.Array
            Single data variable GEOS-Chem output to plot

    Keyword Args (Optional):
        ax: matplotlib axes
            Axes object to plot information
            Default value: None (Will create a new axes)
        plot_type: str
            Either "single_level" or "zonal_mean"
            Default value: "single_level"
        grid: dict
            Dictionary mapping plot_vals to plottable coordinates
            Default value: {} (will attempt to read grid from plot_vals)
        gridtype: str
            "ll" for lat/lon or "cs" for cubed-sphere
            Default value: "" (will automatically determine from grid)
        title: str
            Title to put at top of plot
            Default value: "fill" (will use name attribute of plot_vals
            if available)
        comap: matplotlib Colormap
            Colormap for plotting data values
            Default value: WhGrYlRd
        norm: list
            List with range [0..1] normalizing color range for matplotlib
            methods. Default value: None (will determine from plot_vals)
        unit: str
            Units of plotted data
            Default value: "" (will use units attribute of plot_vals
            if available)
        extent: tuple (minlon, maxlon, minlat, maxlat)
            Describes minimum and maximum latitude and longitude of input
            data.  Default value: None (Will use full extent of plot_vals
            if plot is single level).
        masked_data: numpy array
            Masked area for avoiding near-dateline cubed-sphere plotting
            issues  Default value: None (will attempt to determine from
            plot_vals)
        use_cmap_RdBu: bool
            Set this flag to True to use a blue-white-red colormap
            Default value: False
        log_color_scale: bool
            Set this flag to True to use a log-scale colormap
            Default value: False
        add_cb: bool
```

(continues on next page)

(continued from previous page)

```
            Set this flag to True to add a colorbar to the plot
            Default value: True
        pres_range: list(int)
            Range from minimum to maximum pressure for zonal mean
            plotting. Default value: [0, 2000] (will plot entire
            atmosphere)
        pedge: numpy array
            Edge pressures of vertical grid cells in plot_vals
            for zonal mean plotting.  Default value: np.full((1, 1), -1)
            (will determine automatically)
        pedge_ind: numpy array
            Index of edge pressure values within pressure range in
            plot_vals for zonal mean plotting.
            Default value: np.full((1, 1), -1) (will determine
            automatically)
        log_yaxis: bool
            Set this flag to True to enable log scaling of pressure in
            zonal mean plots.  Default value: False
        xtick_positions: list(float)
            Locations of lat/lon or lon ticks on plot
            Default value: None (will place automatically for
            zonal mean plots)
        xticklabels: list(str)
            Labels for lat/lon ticks
            Default value: None (will determine automatically from
            xtick_positions)
        proj: cartopy projection
            Projection for plotting data
            Default value: ccrs.PlateCarree()
        sg_path: str
            Path to NetCDF file containing stretched-grid info
            (in attributes) for plot_vals.
            Default value: '' (will not be read in)
        ll_plot_func: str
            Function to use for lat/lon single level plotting with
            possible values 'imshow' and 'pcolormesh'. imshow is much
            faster but is slightly displaced when plotting from dateline
            to dateline and/or pole to pole.  Default value: 'imshow'
        vert_params: list(AP, BP) of list-like types
            Hybrid grid parameter A in hPa and B (unitless). Needed if
            grid is not 47 or 72 levels.  Default value: None
        pdfname: str
            File path to save plots as PDF
            Default value: "" (will not create PDF)
        weightsdir: str
            Directory path for storing regridding weights
            Default value: "." (will store regridding files in
            current directory)
        vmin: float
            minimum for colorbars
            Default value: None (will use plot value minimum)
        vmax: float
```

(continues on next page)

(continued from previous page)

```
            maximum for colorbars
            Default value: None (will use plot value maximum)
        return_list_of_plots: bool
            Return plots as a list. This is helpful if you are using
            a cubedsphere grid and would like access to all 6 plots
            Default value: False
        extra_plot_args: various
            Any extra keyword arguments are passed to calls to
            pcolormesh() (CS) or imshow() (Lat/Lon).

    Returns:
        plot: matplotlib plot
            Plot object created from input
    """
```

Function `single_panel()` expects data with a 1-length (or non-existent) `T` (time) dimension, as well as a 1-length or non-existent `Z` (vertical level) dimension.

`single_panel()` contains a few amenities to help with plotting GEOS-Chem data, including automatic grid detection for lat/lon or standard cubed-sphere xarray `DataArray`-s. You can also pass NumPy arrays to plot, though you'll need to manually pass grid info in this case (with the `gridtype`, `pedge`, and `pedge_ind` keyword arguments).

The sample script shown below shows how you can data at a single level and timestep from an `xarray.DataArray` object.

```
#!/usr/bin/env python

import xarray as xr
import matplotlib.pyplot as plt
from gcpy.plot.single_panel import single_panel

# Read data from a file into an xr.Dataset object
dset = xr.open_dataset('GEOSChem.SpeciesConc.20160701_0000z.nc4')

# Extract ozone (v/v) from the xr.Dataset object,
# for time=0 (aka first timestep) and lev=0 (aka surface)
sfc_o3 = dset['SpeciesConcVV_O3'].isel(time=0).isel(lev=0)

# Plot the data!
single_panel(sfc_o3)
plt.show()
```

CHAPTER FIVE

REGRIDDING

GCPy currently supports regridding of data from:

1. GEOS-Chem Classic restart files
2. GEOS-Chem Classic diagnostic files
3. GCHP restart files
4. GCHP diagnostic files
5. HEMCO restart files
6. HEMCO diagnostic files
7. As well as any netCDF file adhering to COARDS or CF conventions.

Regridding is supported across any horizontal resolution and any grid type available in GEOS-Chem, including lat/lon (global or non-global), global standard cubed-sphere, and global stretched-grid. GCPy also supports arbitrary vertical regridding across different vertical resolutions.

Regridding with GCPy is currently undergoing an overhaul. As of the current release, regridding is split into two different categories:

1. Regridding between lat-lon grids using regridding weights computed on the fly by GCPy, and
2. Regridding either lat-lon or cubed-sphere using regridding weights computed as a preprocessing step.

The latter method may be used for creating GCHP standard grid and stretched grid restart files from either GCHP or GEOS-Chem Classic restart files.

5.1 Using Online Regridding Weights

You can regrid existing GEOS-Chem restart or diagnostic files using GCPy function `gcpy.file_regrid`. This function can called directly from the command line (*see the examples below*) or from a Python script or interpreter (`gcpy.file_regrid.file_regrid()`)

Note: For regridding to or from GCHP stretched-grid restart files, we recommend using the *offline regridding weights method*.

The syntax of `file_regrid` is as follows:

```
def file_regrid(
        fin,
        fout,
```

(continues on next page)

(continued from previous page)

```
        dim_format_in,
        dim_format_out,
        cs_res_out=0,
        ll_res_out='0x0',
        sg_params_in=None,
        sg_params_out=None,
        vert_params_out=None,
):
    """
    Regrids an input file to a new horizontal grid specification
    and saves it as a new file.
    """
```

5.1.1 gcpy.file_regrid required arguments:

fin : str

The input filename

fout : str

The output filename (file will be overwritten if it already exists)

dim_format_in : str

Format of the input file's dimensions. Accepted values are:

- `classic`: For GEOS-Chem Classic restart & diagnostic files
- `checkpoint` : For GCHP checkpoint & restart files
- `diagnostic`: For GCHP diagnostic files

dim_format_out : str

Format of the output file's dimensions. Accepted values are:

- `classic`: For GEOS-Chem Classic restart & diagnostic files
- `checkpoint` : For GCHP checkpoint & restart files
- `diagnostic`: For GCHP diagnostic files

5.1.2 gcpy.file_regrid optional arguments:

sg_params_in : list of float

Stretching parameters (`stretch-factor`, `target-longitude`, `target-latitude`) for the input grid. Only needed when the data contained in file *fin* is on a GCHP stretched grid.

Default value: `[1.0, 170.0, -90.0]`

sg_params_out : list of float

Stretching parameters (`stretch-factor`, `target-longitude`, `target-latitude`) for the output grid. Only needed when the data to be contained in file *fout* is to be placed on a GCHP stretched grid.

Default value: `[1.0, 170.0, -90.0]`

cs_res_out : int

Cubed-sphere resolution of the output dataset. Only needed when the data in file *fin* is on a GCHP cubed-sphere grid.

Default value: `0`

ll_res_out : str

The lat/lon resolution of the output dataset. Only needed when the data to be contained in file *fout* is to be placed on a GEOS-Chem Classic lat-lon grid.

Default value: `"0x0"`.

vert_params_out : list of float

Hybrid grid parameter A (in `hPa` and B (`unitless`), returned in list format: `[A, B]`

Default value: `None`

5.1.3 Examples

As stated previously, you can call `gcpy.file_regrid.file_regrid()` from a Python script, or from the command line. Here we shall focus on command-line examples.

1. Regrid a 4x5 GEOS-Chem Classic restart or diagnostic file to a GEOS-Chem Classic 2x2.5 file:

```
$ python -m gcpy.file_regrid               \
  --filein         /path/to/file_4x5.nc4  \
  --dim_format_in  classic                \
  --fileout        /path/to/file_2x25.nc4 \
  --ll_res_out     2x2.5                  \
  --dim_format_out classic
```

2. Regrid a 4x5 GEOS-Chem Classic restart or diagnostic file to a GCHP C24 restart file:

```
$ python -m gcpy.file_regrid               \
  --filein         /path/to/file_4x5.nc4  \
  --dim_format_in  classic                \
  --fileout        /path/to/file_c24.nc4  \
  --cs_res_out     24                     \
  --dim_format_out checkpoint
```

3. Regrid a GCHP C48 restart file to a GCHP stretched grid C48 restart file. The stretch parameters are:
   - stretch-factor: 5
   - target-longitude: -72
   - target-latitude: 41

```
$ python -m gcpy.file_regrid                  \
  --filein         /path/to/file_c48.nc4     \
  --dim_format_in  checkpoint                \
  --fileout        /path/to/file_c48_sg.nc4  \
  --cs_res_out     48                        \
```

(continues on next page)

(continued from previous page)

```
  --dim_format_out checkpoint                \
  --sg_params_out  5 -72 41
```

4. Regrid the GCHP stretched grid C48 restart file from Example 3 above to a GCHP C24 diagnostic file.

```
$ python -m gcpy.file_regrid                 \
  --filein         /path/to/file_c48_sg.nc4  \
  --sg_params_in   5 -72 41                  \
  --dim_format_in  checkpoint                \
  --fileout        /path/to/file_c24.nc4     \
  --cs_res_out     24                        \
  --dim_format_out diagnostic
```

5.2 Using Offline Regridding Weights

This approach requires generating regridding weights using python packages gridspec and sparselt. Regridding with GCPy, `gridspec` and `sparselt` is a three stage process:

1. Create grid specifications for the source and target grids using `gridspec`.
2. Create regridding weights for the transformation using `ESMF_RegridWeightGen`.
3. Run the regridding operation using the `regrid_restart_file` submodule of GCPy.

Note: As of GCPy 1.4.0, the default GCPy environment (aka `gcpy_env`) now contains `gridspec` and `sparselt` packages. You no longer need to use the separate `gchp_regridding` environment as in prior versions.

5.2.1 gcpy.regrid_restart_file required arguments:

There are three arguments required by the GCPy function `regrid_restart_file`:

`file_to_regrid` `: str`
    The GEOS-Chem Classic or GCHP data file to be regridded.

`regridding_weights_file` `: str`
    Regridding weights to be used in the regridding transformation, generated by `ESMF_RegridWeightGen`

`template_file` `: str`
    The GC-Classic or GCHP restart file to use as a template for the regridded restart file. Attributes, dimensions, and variables for the output file will be taken from this template.

5.2.2 gcpy.regrid_restart_file optional arguments:

There are four optional arguments, all of which are for regridded to a stretched cubed-sphere grid.

--stretched-grid : switch

A switch to indicate that the target grid is a stretched cubed-sphere grid.

--stretch-factor : float

The grid stretching factor for the target stretched grid. Only takes effect when `--stretched-grid` is set. See the GCHP documentation for more information. Make sure this value exactly matches the value you plan to use in GCHP configuration file `setCommonRunSettings.sh`.

--target-latitude : float

The latitude of the centre point for stretching the target grid. Only takes effect when `--stretched-grid` is set. See the GCHP documentation for more information. Make sure this value exactly matches the value you plan to use in GCHP configuration file `setCommonRunSettings.sh`.

--target-longitude : float

The longitude of the centre point for stretching the target grid. Only takes effect when `--stretched-grid` is set. See the GCHP documentation for more information. Make sure this value exactly matches the value you plan to use in GCHP configuration file `setCommonRunSettings.sh`.

5.2.3 Example 1: Standard Lat-Lon to Cubed-Sphere Regridding

This example will show regridding a GC-Classic 4x5 restart file to a GCHP c24 restart file.

1. Activate your GCPy environment.

```
$ mamba activate gcpy_env  # Or whatever your environment's name is
```

2. Create a lat-lon source grid specification using `gridspec-create`.

```
$ gridspec-create latlon --pole-centered --half-polar 46 72
```

   This will produce 1 file: `regular_lat_lon_46x72.nc`.

3. Create a target grid specification using `gridspec-create`.

```
$ gridspec-create gcs 24
```

   This will produce 7 files: `c24_gridspec.nc` and `c24.tile[1-6].nc`

4. Create the regridding weights for the regridding transformation (46x72 to C24) using `ESMF_RegridWeightGen`.

```
$ ESMF_RegridWeightGen                    \
  --source      regular_lat_lon_46x72.nc \
  --destination c24_gridspec.nc          \
  --method      conserve                 \
  --weight      46x72_to_c24_weights.nc
```

   This will produce a log file, `PET0.RegridWeightGen.Log`, and our regridding weights, `46x72_to_c24_weights.nc`

5. Use the grid weights produced in previous steps to complete the regridding.

```
$ python -m gcpy.regrid_restart_file          \
  GEOSChem.Restart.20190701_0000z.nc4         \
  46x72_to_c24_weights.nc                     \
  GEOSChem.Restart.20190701_0000z.c24_old.nc4
```

The arguments to `gcpy.regrid_restart_file` *are described above*. In this example (lat-lon to cubed-sphere) we need to use a GEOS-Chem Classic restart file as the file to be regridded and a GCHP restart file as the template file.

Note: The resolution of the template file does not matter as long as it contains all of the variables and attributes that you wish to include in the regridded restart file.

After running `gcpy.regrid_restart_file`, a single restart file named `new_restart_file.nc` will be created. You can rename this file and use it to initialize your GCHP C24 simulation.

6. Deactivate your GCPy environment when finished.

```
$ mamba deactivate
```

5.2.4 Example 2: Standard Cubed-Sphere to Cubed-Sphere Regridding

We will use the example of regridding the out-of-the-box `GEOSChem.Restart.20190701_0000z.c48.nc4` restart file from C48 to C60 to demonstrate the standard cubed-sphere regridding process:

1. Activate your GCPy environment.

```
$ mamba activate gcpy_env  # Or whatever your environment's name is
```

2. Create a source grid specification using `gridspec-create`.

```
$ gridspec-create gcs 48
```

This will produce 7 files: `c48_gridspec.nc` and `c48.tile[1-6].nc`

3. Create a target grid specification using `gridspec-create`.

```
$ gridspec-create gcs 60
```

Again, this will produce 7 files: `c60_gridspec.nc` and `c60.tile[1-6].nc`

4. Create the regridding weights for the regridding transformation (C48 to C60) using `ESMF_RegridWeightGen`.

```
$ ESMF_RegridWeightGen                  \
  --source      c48_gridspec.nc         \
  --destination c60_gridspec.nc         \
  --method      conserve                \
  --weight      c48_to_c60_weights.nc
```

This will produce a log file, `PET0.RegridWeightGen.Log`, and our regridding weights, `c48_to_c60_weights.nc`

5. Use the grid weights produced in earlier steps to complete the regridding.

```
$ python -m gcpy.regrid_restart_file      \
  GEOSChem.Restart.20190701_0000z.c48.nc4 \
  c48_to_c60_weights.nc                   \
  GEOSChem.Restart.20190701_0000z.c48.nc4
```

The arguments to `gcpy.regrid_restart_file` *are described above*. Because we are regridding from one cubed-sphere grid to another cubed-sphere grid, we can use the file to be regridded as the template file.

After running `gcpy.regrid_restart_file`, a single restart file named `new_restart_file.nc` will be created. You can rename this file as you wish and use it for your GCHP C60 simulation.

6. Deactivate your GCPy environment when you have finished.

```
$ mamba deactivate
```

5.2.5 Example 3: Standard to Stretched Cubed-Sphere Regridding

This example regrids the out-of-the-box c48 restart file (`GEOSChem.Restart.20190701_0000z.c48.nc4`) from a standard cubed-sphere grid to a stretched grid. The base resolution will remain the same at c48. The regridded file will have a stretch factor of 4.0 over Bermuda which means a regional grid resolution of c196 (4 times 48) in that area.

1. Activate your GCPy environment:

```
$ mamba activate gcpy_env  # Or whatever your environment's name is
```

2. Create a source grid specification using `gridspec-create`.

```
$ gridspec-create gcs 48
```

This will produce 7 files: `c48_gridspec.nc` and `c48.tile[1-6].nc`

3. Create a target grid specification using `gridspec-create`. This will be for the stretched grid.

```
$ gridspec-create sgcs 48 -s 4.0 -t 32.0 -64.0
```

Here, the `-s` option denotes the stretch factor and the `-t` option denotes the latitude / longitude of the centre point of the grid stretch.

Again, this will produce 7 files: `c48_..._gridspec.nc` and `c48_..._tile[1-6].nc`, where `...` denotes randomly generated characters. Be sure to look for these since you will need them in the next step.

4. Create the regridding weights for the regridding transformation (C48 to C48-stretched) using `ESMF_RegridWeightGen`, replacing `c48_..._gridspec.nc` with the actual name of the file created in the previous step. An example is shown below.

```
$ ESMF_RegridWeightGen                              \
  --source      c48_gridspec.nc                     \
  --destination c48_s4d00_tdtdqp9ktebm5_gridspec.nc \
  --method      conserve                            \
  --weight      c48_to_c48_stretched_weights.nc
```

This will produce a log file, `PET0.RegridWeightGen.Log`, and our regridding weights, `c48_to_c48_stretched_weights.nc`

5. Use the grid weights produced in earlier steps to complete the regridding.

```
$ python -m gcpy.regrid_restart_file       \
   --stretched-grid                        \
   --stretch-factor 4.0                    \
   --target-latitude 32.0                  \
   --target-longitude -64.0                \
   GEOSChem.Restart.20190701_0000z.c48.nc4 \
   c48_to_c48_stretched_weights.nc         \
   GEOSChem.Restart.20190701_0000z.c48.nc4
```

The arguments to `gcpy.regrid_restart_file` *are described above*. Because we are regridding from one cubed-sphere grid to another cubed-sphere grid, we can use the file to be regridded as the template file.

This will produce a single file, `new_restart_file.nc`, regridded from C48 standard to C48 stretched with a stretch factor of 4.0 over 32.0N, -64.0E, that you can rename and use as you please.

Tip: It is generally a good idea to rename the file to include the grid resolution, stretch factor, and target lat/lon for easy reference. You can copy it somewhere to keep long-term and link to it from the GCHP Restarts subdirectory in the run directory.

```
$ mv new_restart_file.nc GEOSChem.Restart.20190701_0000z.c120.s4_32N_64E.nc
```

You can also easily reference the file's stretch parameters by looking at the global attributes in the file. When using the file as a restart file in GCHP make sure that you use the exact same parameters in both the file's global attributes and GCHP configuration file `setCommonRunSettings.sh`.

6. Deactivate your GCPy environment when you have finished.

```
$ mamba deactivate
```

5.3 Regridding for Plotting in GCPy

When plotting in GCPy (e.g. through `gcpy.compare_single_level()` or `gcpy.compare_zonal_mean()`), the vast majority of regridding is handled internally. You can optionally request a specific horizontal comparison resolution in `compare_single_level()` and `compare_zonal_mean()`. Note that all regridding in these plotting functions only applies to the comparison panels (not the top two panels which show data directly from each dataset). There are only two scenarios where you will need to pass extra information to GCPy to help it determine grids and to regrid when plotting.

5.3.1 Pass stretched-grid file paths

Stretched-grid parameters cannot currently be automatically determined from grid coordinates. If you are plotting stretched-grid data in `gcpy.compare_single_level()` or `gcpy.compare_zonal_mean()` (even if regridding to another format), you need to use the `sg_ref_path` or `sg_dev_path` arguments to pass the path of your original stretched-grid restart file to GCPy. If using `single_panel()`, pass the file path using `sg_path`. Stretched-grid restart files created using GCPy contain the specified stretch factor, target longitude, and target latitude in their metadata. Currently, output files from stretched-grid runs of GCHP do not contain any metadata that specifies the stretched-grid used.

5.3.2 Pass vertical grid parameters for non-72/47-level grids

GCPy automatically handles regridding between different vertical grids when plotting except when you pass a dataset that is not on the typical 72-level or 47-level vertical grids. If using a different vertical grid, you will need to pass the corresponding grid parameters using the `ref_vert_params` or `dev_vert_params` keyword arguments.

5.3.3 Automatic regridding decision process

When you do not specify a horizontal comparison resolution using the `cmpres` argument in `gcpy.compare_single_level()` and `compare_zonal_mean()`, GCPy follows several steps to determine what comparison resolution it should use:

- If both input grids are lat/lon, use the highest resolution between them (don't regrid if they are the same resolution).
- Else if one grid is lat/lon and the other is cubed-sphere (standard or stretched-grid), use a 1x1.25 lat/lon grid.
- Else if both grids are cubed-sphere and you are plotting zonal means, use a 1x1.25 lat/lon grid.
- Else if both grids are standard cubed-sphere, use the highest resolution between them (don't regrid if they are the same resolution).
- Else if one or more grids is a stretched-grid, use the grid of the ref dataset.

For differing vertical grids, the smaller vertical grid is currently used for comparisons.

CHAPTER SIX

BENCHMARKING

The GEOS-Chem Support Team uses GCPy to produce comparison plots and summary tables from GEOS-Chem benchmark simulations. In this chapter we will describe this capability of GCPy.

6.1 Location of benchmark plotting scripts

The source code for creating benchmark plots is located in the `gcpy/benchmark` directory tree.

Table 1: **Contents of the gcpy/benchmark directory**

File or folder	Description
`run_benchmark.py`	Benchmark driver script
`benchmark_slurm.sh`	Bash script to submit `run_benchmark,py` as a SLURM batch job
`cloud/`	Directory containing template config files (in YAML format) for 1-hour and 1-month benchmark plot jobs on the AWS cloud.
`config/`	Directory containing editable config files (in YAML format) for 1-month and 1-year benchmark plot jobs.
`__init__.py`	Python import script
`modules/`	Contains Python modules imported into the `run_benchmark.py` script.
`README.md`	Readme file in Markdown format

Note: As of this writing, the benchmarking scripts still use several *plotting* and *tabling* functions from module `gcpy.benchmark_funcs`. We are currently in the process of moving the functions contained in `gcpy.benchmark_funcs` to the `gcpy/benchmark/modules` directory.

6.2 Steps to create benchmark plots

Follow these instructions to create comparison plots and summary tables from GEOS-Chem benchmark simulations.

1. Copy a configuration file from the `gcpy/benchmark/config` directory.

   In this example we will use the configuration file that will create plots from 1-year full-chemistry benchmark simulations. (Configuration files for other benchmark types have a similar layout.)

```
$ cp /path/to/GCPy/gcpy/benchmark/config/1yr_fullchem_benchmark.yml .
```

2. Edit the paths section of the configuration file to specify the proper directory paths for your system.

```
# Configuration for 1-year FullChemBenchmark
#
# paths:
#   main_dir:     High-level directory containing ref & dev rundirs
#   results_dir:  Directory where plots/tables will be created
#   weights_dir:  Path to regridding weights
#   spcdb_dir:    Folder in which the species_database.yml file is
#                  located.  If set to "default", then will look for
#                  species_database.yml in one of the Dev rundirs.
#   obs_data_dir: Path to observational data (for models vs obs plots)
#
paths:
  main_dir: /path/to/benchmark/main/dir    # EDIT AS NEEDED
  results_dir: /path/to/BenchmarkResults   # EDIT AS NEEDED
  weights_dir: /n/holyscratch01/external_repos/GEOS-CHEM/gcgrid/data/ExtData/GCHP/
↪RegriddingWeights
  spcdb_dir: default
  obs_data_dir: /path/to/observational/data
```

3. Edit the data section to specify the directories (and labels) for the Ref and Dev versions for GEOS-Chem Classic and GCHP.

```
#
# data: Contains configurations for ref and dev runs
#   version:         Version string (must not contain spaces)
#   dir:             Path to run directory
#   outputs_subdir:  Subdirectory w/ GEOS-Chem diagnostic files
#   restarts_subdir: Subdirectory w/ GEOS-Chem restarts
#   bmk_start:       Simulation start date (YYYY-MM-DDThh:mm:ss)
#   bmk_end:         Simulation end date (YYYY-MM-DDThh:mm:ss)
#   resolution:      GCHP resolution string
#
data:
  ref:
    gcc:
      version: GCC_ref
      dir: GCC_ref
      outputs_subdir: OutputDir
      restarts_subdir: Restarts
      bmk_start: "2019-01-01T00:00:00"
      bmk_end: "2020-01-01T00:00:00"
    gchp:
      version: GCC_dev
      dir: GCC_dev
      outputs_subdir: OutputDir
      restarts_subdir: Restarts
```

(continues on next page)

(continued from previous page)

```
      bmk_start: "2019-01-01T00:00:00"
      bmk_end: "2020-01-01T00:00:00"
      is_pre_14.0: False
      resolution: c24
  dev:
    gcc:
      version: GCC_dev
      dir: GCC_dev
      outputs_subdir: OutputDir
      restarts_subdir: Restarts
      bmk_start: "2019-01-01T00:00:00"
      bmk_end: "2020-01-01T00:00:00"
    gchp:
      version: GCC_dev
      dir: GCC_dev
      restarts_subdir: Restarts
      bmk_start: "2019-01-01T00:00:00"
      bmk_end: "2020-01-01T00:00:00"
      is_pre_14.0: False
      resolution: c24
```

4. Edit the comparisons section to specify the types of comparisons you would like to perform.

```
#
# comparisons: Specifies the comparisons to perform.
#
comparisons:
  gcc_vs_gcc:
    run: True
    dir: GCC_version_comparison
    tables_subdir: Tables
  gchp_vs_gcc:
    run: True
    dir: GCHP_GCC_comparison
    tables_subdir: Tables
  gchp_vs_gchp:
    run: True
    dir: GCHP_version_comparison
    tables_subdir: Tables
  gchp_vs_gcc_diff_of_diffs:
    run: True
    dir: GCHP_GCC_diff_of_diffs
```

5. Edit the outputs section to select the plots and tables that you would like to generate.

```
#
# outputs: Specifies the plots and tables to generate
#
outputs:
   plot_conc: True
   plot_emis: True
```

(continues on next page)

(continued from previous page)

```
    emis_table: True
    plot_jvalues: True
    plot_aod: True
    mass_table: True
    ops_budget_table: False
    aer_budget_table: True
    Ox_budget_table: True
    ste_table: True # GCC only
    OH_metrics: True
    plot_models_vs_obs: True
    plot_options:
      by_spc_cat: True
      by_hco_cat: True
```

6. Edit the `n_cores` setting if you wish to change the number of computational cores to use. If not, leave `n_cores` set to `-1`, which will use as many cores as possible.

```
#
# n_cores: Specify the number of cores to use.
# -1: Use $OMP_NUM_THREADS         cores
# -2: Use $OMP_NUM_THREADS - 1     cores
# -N: Use $OMP_NUM_THREADS - (N-1) cores
#  1: Disable parallelization (use a single core)
#
n_cores: -1
```

7. Run the `run.benchmark.py` script. You may do this in 2 ways:

   1. Direct execution from the command line:

   ```
   (gcpy_env) $ python -m gcpy.benchmark.run_benchmark
   1yr_fullchem_benchmark.yml
   ```

   2. Batch execution with the SLURM scheduler. First, copy the `benchmark_slurm.sh` script to your current directory:

   ```
   (gcpy_env) $ cp /path/to/GCPy/gcpy/benchmark/benchmark_slurm.sh .
   ```

   Next, edit your local copy of `benchmark_slurm.sh` to specify your SLURM partition name, number of cores, the name of your Python environment and the configuration file to use.

   ```
   #!/bin/bash

   #SBATCH -c 8
   #SBATCH -N 1
   #SBATCH -t 0-4:00
   #SBATCH -p seas_compute,shared
   #SBATCH --mem=100000
   #SBATCH --mail-type=END
   ```

(continues on next page)

(continued from previous page)

```
#==============================================================================
# This us a sample SLURM script that you can use to run the GCPy
# benchmark plotting code as a SLURM batch job.
#
# You can modify the SLURM parameters above for your setup.
#
# Tip: Using less cores can reduce the amount of memory required.
#==============================================================================

# Apply all bash initialization settings
. ~/.bashrc

# Make sure to set multiple threads; Joblib will use multiple
# cores to parallelize certain plotting operations.
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export OMP_STACKSIZE=500m

# Turn on Python environment (edit for your setup)
mamba activate gcpy_env

# Specify a YAML file with benchmark options
# Uncomment the file that you wish:
#config="1mo_benchmark.yml"
config="1yr_fullchem_benchmark.yml"
#config="1yr_tt_benchmark.yml"

# Call the run_benchmark script to make the plots
python -m gcpy.benchmark.run_benchmark "${config}" > benchmark.log 2>&1

# Turn off python environment
mamba deactivate

exit 0
```

Lastly, start the SLURM batch execution with this command:

```
$ sbatch benchmark_slurm.sh
```

6.3 Benchmark plotting functions

Module `gcpy.benchmark_funcs` contains several functions for creating plots and tables from GEOS-Chem benchmark simulations. The specific outputs generated have been requested by the GEOS-Chem Steering Committee in order to facilitate comparing benchmark output from different model versions.

In this section, we will describe functions that create comparison plots from GEOS-Chem benchmark simulation output. The functions to create summary tables will be described *in a separate section*.

Note: We are working towards moving all benchmark-related source code to the `gcpy/benchmark/` directory tree. For the time being, the `benchmark_funcs.py` script is located in the `/path/to/GCPy/gcpy/` directory.

Table 2: **Functions creating comparison plots from benchmark simulation output**

Function	Type of 6-panel comparison plot created
`make_benchmark_aod_plots()`	Comparison plots for aerosol optical depth
`make_benchmark_conc_plots()`	Species concentration
`make_benchmark_emis_plots()`	Emissions (by species and catgegory)
`make_benchmark_jvalue_plots()`	Comparison plots for J-values (photolysis)
`make_benchmark_wetdep_plots()`	Comparison plots for species wet deposition

The functions listed above create comparison plots of most GEOS-Chem output variables divided into specific categories, e.g. species categories such as `Aerosols` or `Bromine` for the `SpeciesConcVV` diagnostic. In eachcategory, these function create single level PDFs for the surface and 500hPa and zonal mean PDFs for the entire atmosphere and only the stratosphere (defined a 1-100hPa). For `make_benchmark_emis_plots()`, only single level plots at the surface are produced. All of these plotting functions include bookmarks within the generated PDFs that point to the pages containing each plotted quantity. Thus these functions serve as tools for quickly creating comprehensive plots comparing two GEOS-Chem runs. These functions are used to create the publicly available plots for 1-month and 1-year benchmarks of new versions of GEOS-Chem.

Many of the plotting functions listed above use pre-defined lists of variables in YAML files. If one dataset includes a variable but the other dataset does not, the data for that variable in the latter dataset will be considered to be NaN and will be plotted as such.

6.3.1 make_benchmark_aod_plots

This function creates column optical depth plots using the Aerosols diagnostic output.

```
def make_benchmark_aod_plots(
        ref,
        refstr,
        dev,
        devstr,
        varlist=None,
        dst="./benchmark",
        subdst=None,
        cmpres=None,
        overwrite=False,
        verbose=False,
        log_color_scale=False,
        sigdiff_files=None,
        weightsdir='.',
        n_job=-1,
        time_mean=False,
        spcdb_dir=os.path.dirname(__file__)
):
    """
    Creates PDF files containing plots of column aerosol optical
    depths (AODs) for model benchmarking purposes.

    Args:
        ref: str
            Path name for the "Ref" (aka "Reference") data set.
```

(continues on next page)

(continued from previous page)

```
        refstr: str
            A string to describe ref (e.g. version number)
        dev: str
            Path name for the "Dev" (aka "Development") data set.
            This data set will be compared against the "Reference"
            data set.
        devstr: str
            A string to describe dev (e.g. version number)

    Keyword Args (optional):
        varlist: list of str
            List of AOD variables to plot.  If not passed, then all
            AOD variables common to both Dev and Ref will be plotted.
            Use the varlist argument to restrict the number of
            variables plotted to the pdf file when debugging.
            Default value: None
        dst: str
            A string denoting the destination folder where a
            PDF file  containing plots will be written.
            Default value: ./benchmark.
        subdst: str
            A string denoting the sub-directory of dst where PDF
            files containing plots will be written.  In practice,
            subdst is only needed for the 1-year benchmark output,
            and denotes a date string (such as "Jan2016") that
            corresponds to the month that is being plotted.
            Default value: None
        cmpres: string
            Grid resolution at which to compare ref and dev data, e.g. '1x1.25'
        overwrite: bool
            Set this flag to True to overwrite files in the
            destination folder (specified by the dst argument).
            Default value: False.
        verbose: bool
            Set this flag to True to print extra informational output.
            Default value: False
        log_color_scale: bool
            Set this flag to True to enable plotting data (not diffs)
            on a log color scale.
            Default value: False
        sigdiff_files: list of str
            Filenames that will contain the list of quantities having
            having significant differences in the column AOD plots.
            These lists are needed in order to fill out the benchmark
            approval forms.
            Default value: None
        weightsdir: str
            Directory in which to place (and possibly reuse) xESMF regridder
            netCDF files.
            Default value: '.'
        n_job: int
            Defines the number of simultaneous workers for parallel plotting.
```

(continues on next page)

(continued from previous page)

```
            Set to 1 to disable parallel plotting. Value of -1 allows the
            application to decide.
            Default value: -1
        spcdb_dir: str
            Directory of species_datbase.yml file
            Default value: Directory of GCPy code repository
        time_mean : bool
            Determines if we should average the datasets over time
            Default value: False
    """
```

6.3.2 make_benchmark_conc_plots

This function creates species concentration plots using the SpeciesConc diagnostic output by default. In particular:

- This function is the only benchmark plotting function that supports diff-of-diffs plotting, in which 4 datasets are passed and the differences between two groups of `Ref` datasets vs. two groups of `Dev` datasets is plotted (typically used for comparing changes in GCHP vs. changes in GEOS-Chem Classic across model versions).
- This is also the only benchmark plotting function that sends plots to separate folders based on category (as denoted by the plot_by_spc_cat flag). The full list of species categories is denoted in benchmark_categories.yml (included in GCPy).
- In this function, parallelization occurs at the species category level. In all other functions, parallelization occurs within calls to `compare_single_level()` and `compare_zonal_mean()`.=

```
def make_benchmark_conc_plots(
        ref,
        refstr,
        dev,
        devstr,
        dst="./benchmark",
        subdst=None,
        overwrite=False,
        verbose=False,
        collection="SpeciesConc",
        benchmark_type="FullChemBenchmark",
        cmpres=None,
        plot_by_spc_cat=True,
        restrict_cats=[],
        plots=["sfc", "500hpa", "zonalmean"],
        use_cmap_RdBu=False,
        log_color_scale=False,
        sigdiff_files=None,
        normalize_by_area=False,
        cats_in_ugm3=["Aerosols", "Secondary_Organic_Aerosols"],
        areas=None,
        refmet=None,
        devmet=None,
        weightsdir='.',
        n_job=-1,
        second_ref=None,
```

(continues on next page)

(continued from previous page)

```
        second_dev=None,
        time_mean=False,
        spcdb_dir=os.path.dirname(__file__)
):
    """
    Creates PDF files containing plots of species concentration
    for model benchmarking purposes.

    Args:
        ref: str
            Path name for the "Ref" (aka "Reference") data set.
        refstr: str
            A string to describe ref (e.g. version number)
        dev: str
            Path name for the "Dev" (aka "Development") data set.
            This data set will be compared against the "Reference"
            data set.
        devstr: str
            A string to describe dev (e.g. version number)

    Keyword Args (optional):
        dst: str
            A string denoting the destination folder where a PDF
            file containing plots will be written.
            Default value: ./benchmark
        subdst: str
            A string denoting the sub-directory of dst where PDF
            files containing plots will be written.  In practice,
            subdst is only needed for the 1-year benchmark output,
            and denotes a date string (such as "Jan2016") that
            corresponds to the month that is being plotted.
            Default value: None
        overwrite: bool
            Set this flag to True to overwrite files in the
            destination folder (specified by the dst argument).
            Default value: False
        verbose: bool
            Set this flag to True to print extra informational output.
            Default value: False
        collection: str
            Name of collection to use for plotting.
            Default value: "SpeciesConc"
        benchmark_type: str
            A string denoting the type of benchmark output to plot, options are
            FullChemBenchmark, TransportTracersBenchmark, or CH4Benchmark.
            Default value: "FullChemBenchmark"
        cmpres: string
            Grid resolution at which to compare ref and dev data, e.g. '1x1.25'
        plot_by_spc_cat: logical
            Set this flag to False to send plots to one file rather
            than separate file per category.
            Default value: True
```

(continues on next page)

(continued from previous page)

```
        restrict_cats: list of strings
            List of benchmark categories in benchmark_categories.yml to make
            plots for. If empty, plots are made for all categories.
            Default value: empty
        plots: list of strings
            List of plot types to create.
            Default value: ['sfc', '500hpa', 'zonalmean']
        log_color_scale: bool
            Set this flag to True to enable plotting data (not diffs)
            on a log color scale.
            Default value: False
        normalize_by_area: bool
            Set this flag to true to enable normalization of data
            by surfacea area (i.e. kg s-1 --> kg s-1 m-2).
            Default value: False
        cats_in_ugm3: list of str
            List of benchmark categories to to convert to ug/m3
            Default value: ["Aerosols", "Secondary_Organic_Aerosols"]
        areas: dict of xarray DataArray:
            Grid box surface areas in m2 on Ref and Dev grids.
            Default value: None
        refmet: str
            Path name for ref meteorology
            Default value: None
        devmet: str
            Path name for dev meteorology
            Default value: None
        sigdiff_files: list of str
            Filenames that will contain the lists of species having
            significant differences in the 'sfc', '500hpa', and
            'zonalmean' plots.  These lists are needed in order to
            fill out the benchmark approval forms.
            Default value: None
        weightsdir: str
            Directory in which to place (and possibly reuse) xESMF regridder
            netCDF files.
            Default value: '.'
        n_job: int
            Defines the number of simultaneous workers for parallel plotting.
            Set to 1 to disable parallel plotting. Value of -1 allows the
            application to decide.
            Default value: -1
        second_ref: str
            Path name for a second "Ref" (aka "Reference") data set for
            diff-of-diffs plotting. This dataset should have the same model
            type and grid as ref.
            Default value: None
        second_dev: str
            Path name for a second "Ref" (aka "Reference") data set for
            diff-of-diffs plotting. This dataset should have the same model
            type and grid as ref.
            Default value: None
```

(continues on next page)

(continued from previous page)

```
        spcdb_dir: str
            Directory of species_datbase.yml file
            Default value: Directory of GCPy code repository
        time_mean : bool
            Determines if we should average the datasets over time
            Default value: False
    """
```

6.3.3 make_benchmark_emis_plots

This function generates plots of total emissions using output from `HEMCO_diagnostics.*` (for GEOS-Chem Classic) and/or `GCHP.Emissions.*` output files.

```
def make_benchmark_emis_plots(
        ref,
        refstr,
        dev,
        devstr,
        dst="./benchmark",
        subdst=None,
        plot_by_spc_cat=False,
        plot_by_hco_cat=False,
        benchmark_type="FullChemBenchmark",
        cmpres=None,
        overwrite=False,
        verbose=False,
        flip_ref=False,
        flip_dev=False,
        log_color_scale=False,
        sigdiff_files=None,
        weightsdir='.',
        n_job=-1,
        time_mean=False,
        spcdb_dir=os.path.dirname(__file__)
):
    """
    Creates PDF files containing plots of emissions for model
    benchmarking purposes. This function is compatible with benchmark
    simulation output only. It is not compatible with transport tracers
    emissions diagnostics.

    Args:
        ref: str
            Path name for the "Ref" (aka "Reference") data set.
        refstr: str
            A string to describe ref (e.g. version number)
        dev: str
            Path name for the "Dev" (aka "Development") data set.
            This data set will be compared against the "Reference"
            data set.
        devstr: str
```

(continues on next page)

(continued from previous page)

```
            A string to describe dev (e.g. version number)

    Keyword Args (optional):
        dst: str
            A string denoting the destination folder where
            PDF files containing plots will be written.
            Default value: './benchmark
        subdst: str
            A string denoting the sub-directory of dst where PDF
            files containing plots will be written.  In practice,
            and denotes a date string (such as "Jan2016") that
            corresponds to the month that is being plotted.
            Default value: None
        plot_by_spc_cat: bool
            Set this flag to True to separate plots into PDF files
            according to the benchmark species categories (e.g. Oxidants,
            Aerosols, Nitrogen, etc.)  These categories are specified
            in the YAML file benchmark_species.yml.
            Default value: False
        plot_by_hco_cat: bool
            Set this flag to True to separate plots into PDF files
            according to HEMCO emissions categories (e.g. Anthro,
            Aircraft, Bioburn, etc.)
            Default value: False
        benchmark_type: str
            A string denoting the type of benchmark output to plot, options are
            FullChemBenchmark, TransportTracersBenchmark, or CH4Benchmark.
            Default value: "FullChemBenchmark"
        cmpres: string
            Grid resolution at which to compare ref and dev data, e.g. '1x1.25'
        overwrite: bool
            Set this flag to True to overwrite files in the
            destination folder (specified by the dst argument).
            Default value: False
        verbose: bool
            Set this flag to True to print extra informational output.
            Default value: False
        flip_ref: bool
            Set this flag to True to reverse the vertical level
            ordering in the "Ref" dataset (in case "Ref" starts
            from the top of atmosphere instead of the surface).
            Default value: False
        flip_dev: bool
            Set this flag to True to reverse the vertical level
            ordering in the "Dev" dataset (in case "Dev" starts
            from the top of atmosphere instead of the surface).
            Default value: False
        log_color_scale: bool
            Set this flag to True to enable plotting data (not diffs)
            on a log color scale.
            Default value: False
         sigdiff_files: list of str
```

(continues on next page)

(continued from previous page)

```
            Filenames that will contain the lists of species having
            significant differences in the 'sfc', '500hpa', and
            'zonalmean' plots.  These lists are needed in order to
            fill out the benchmark approval forms.
            Default value: None
        weightsdir: str
            Directory in which to place (and possibly reuse) xESMF regridder
            netCDF files.
            Default value: '.'
        n_job: int
            Defines the number of simultaneous workers for parallel plotting.
            Set to 1 to disable parallel plotting.
            Value of -1 allows the application to decide.
            Default value: -1
        spcdb_dir: str
            Directory of species_datbase.yml file
            Default value: Directory of GCPy code repository
        time_mean : bool
            Determines if we should average the datasets over time
            Default value: False

    Remarks:
        (1) If both plot_by_spc_cat and plot_by_hco_cat are
            False, then all emission plots will be placed into the
            same PDF file.

        (2) Emissions that are 3-dimensional will be plotted as
            column sums.
           column sums.
"""
```

6.3.4 make_benchmark_jvalue_plots

This function generates plots of J-values using the `JValues` GEOS-Chem output files.

```
def make_benchmark_jvalue_plots(
        ref,
        refstr,
        dev,
        devstr,
        varlist=None,
        dst="./benchmark",
        subdst=None,
        local_noon_jvalues=False,
        cmpres=None,
        plots=["sfc", "500hpa", "zonalmean"],
        overwrite=False,
        verbose=False,
        flip_ref=False,
        flip_dev=False,
        log_color_scale=False,
```

(continues on next page)

(continued from previous page)

```
        sigdiff_files=None,
        weightsdir='.',
        n_job=-1,
        time_mean=False,
        spcdb_dir=os.path.dirname(__file__)
):
    """
    Creates PDF files containing plots of J-values for model
    benchmarking purposes.

    Args:
        ref: str
            Path name for the "Ref" (aka "Reference") data set.
        refstr: str
            A string to describe ref (e.g. version number)
        dev: str
            Path name for the "Dev" (aka "Development") data set.
            This data set will be compared against the "Reference"
            data set.
        devstr: str
            A string to describe dev (e.g. version number)

    Keyword Args (optional):
        varlist: list of str
            List of J-value variables to plot.  If not passed,
            then all J-value variables common to both dev
            and ref will be plotted.  The varlist argument can be
            a useful way of restricting the number of variables
            plotted to the pdf file when debugging.
            Default value: None
        dst: str
            A string denoting the destination folder where a
            PDF file  containing plots will be written.
            Default value: ./benchmark.
        subdst: str
            A string denoting the sub-directory of dst where PDF
            files containing plots will be written.  In practice,
            subdst is only needed for the 1-year benchmark output,
            and denotes a date string (such as "Jan2016") that
            corresponds to the month that is being plotted.
            Default value: None
        local_noon_jvalues: bool
            Set this flag to plot local noon J-values.  This will
            divide all J-value variables by the JNoonFrac counter,
            which is the fraction of the time that it was local noon
            at each location.
            Default value: False
        cmpres: string
            Grid resolution at which to compare ref and dev data, e.g. '1x1.25'
        plots: list of strings
            List of plot types to create.
            Default value: ['sfc', '500hpa', 'zonalmean']
```

(continues on next page)

(continued from previous page)

```
        overwrite: bool
            Set this flag to True to overwrite files in the
            destination folder (specified by the dst argument).
            Default value: False.
        verbose: bool
            Set this flag to True to print extra informational output.
            Default value: False
        flip_ref: bool
            Set this flag to True to reverse the vertical level
            ordering in the "Ref" dataset (in case "Ref" starts
            from the top of atmosphere instead of the surface).
            Default value: False
        flip_dev: bool
            Set this flag to True to reverse the vertical level
            ordering in the "Dev" dataset (in case "Dev" starts
            from the top of atmosphere instead of the surface).
            Default value: False
        log_color_scale: bool
            Set this flag to True if you wish to enable plotting data
            (not diffs) on a log color scale.
            Default value: False
        sigdiff_files: list of str
            Filenames that will contain the lists of J-values having
            significant differences in the 'sfc', '500hpa', and
            'zonalmean' plots.  These lists are needed in order to
            fill out the benchmark approval forms.
            Default value: None
        weightsdir: str
            Directory in which to place (and possibly reuse) xESMF regridder
            netCDF files.
            Default value: '.'
        n_job: int
            Defines the number of simultaneous workers for parallel plotting.
            Set to 1 to disable parallel plotting. Value of -1 allows the
            application to decide.
            Default value: -1
        spcdb_dir: str
            Directory of species_datbase.yml file
            Default value: Directory of GCPy code repository
        time_mean : bool
            Determines if we should average the datasets over time
            Default value: False

    Remarks:
         Will create 4 files containing J-value plots:
            (1 ) Surface values
            (2 ) 500 hPa values
            (3a) Full-column zonal mean values.
            (3b) Stratospheric zonal mean values
         These can be toggled on/off with the plots keyword argument.

         At present, we do not yet have the capability to split the
```

(continues on next page)

(continued from previous page)

```
        plots up into separate files per category (e.g. Oxidants,
        Aerosols, etc.).  This is primarily due to the fact that
        we archive J-values from GEOS-Chem for individual species
        but not family species.  We could attempt to add this
        functionality later if there is sufficient demand.
    """
```

6.3.5 make_benchmark_wetdep_plots

This function generates plots of wet deposition using `WetLossConv` and `WetLossLS` GEOS-Chem output files. It is currently primarily used for 1-Year Transport Tracer benchmarks, plotting values for the following species as defined in benchmark_categories.yml (included in GCPY).

```
def make_benchmark_wetdep_plots(
        ref,
        refstr,
        dev,
        devstr,
        collection,
        dst="./benchmark",
        cmpres=None,
        datestr=None,
        overwrite=False,
        verbose=False,
        benchmark_type="TransportTracersBenchmark",
        plots=["sfc", "500hpa", "zonalmean"],
        log_color_scale=False,
        normalize_by_area=False,
        areas=None,
        refmet=None,
        devmet=None,
        weightsdir='.',
        n_job=-1,
        time_mean=False,
        spcdb_dir=os.path.dirname(__file__)
):
    """
    Creates PDF files containing plots of species concentration
    for model benchmarking purposes.

    Args:
        ref: str
            Path name for the "Ref" (aka "Reference") data set.
        refstr: str
            A string to describe ref (e.g. version number)
        dev: str
            Path name for the "Dev" (aka "Development") data set.
            This data set will be compared against the "Reference"
            data set.
        devstr: str
            A string to describe dev (e.g. version number)
```

(continues on next page)

(continued from previous page)

```
        collection: str
            String name of collection to plot comparisons for.

    Keyword Args (optional):
        dst: str
            A string denoting the destination folder where a PDF
            file containing plots will be written.
            Default value: ./benchmark
        datestr: str
            A string with date information to be included in both the
            plot pdf filename and as a destination folder subdirectory
            for writing plots
            Default value: None
        benchmark_type: str
            A string denoting the type of benchmark output to plot, options are
            FullChemBenchmark, TransportTracersBenchmark, or CH4Benchmark.
            Default value: "FullChemBenchmark"
        overwrite: bool
            Set this flag to True to overwrite files in the
            destination folder (specified by the dst argument).
            Default value: False.
        verbose: bool
            Set this flag to True to print extra informational output.
            Default value: False.
        plots: list of strings
            List of plot types to create.
            Default value: ['sfc', '500hpa', 'zonalmean']
        normalize_by_area: bool
            Set this flag to true to enable normalization of data
            by surfacea area (i.e. kg s-1 --> kg s-1 m-2).
            Default value: False
        areas: dict of xarray DataArray:
            Grid box surface areas in m2 on Ref and Dev grids.
            Default value: None
        refmet: str
            Path name for ref meteorology
            Default value: None
        devmet: str
            Path name for dev meteorology
            Default value: None
        n_job: int
            Defines the number of simultaneous workers for parallel plotting.
            Set to 1 to disable parallel plotting. Value of -1 allows the
            application to decide.
            Default value: -1
        spcdb_dir: str
            Directory of species_datbase.yml file
            Default value: Directory of GCPy code repository
        time_mean : bool
            Determines if we should average the datasets over time
            Default value: False
    """
```

6.4 Benchmark tabling functions

Table 3: **Functions creating summary tables from benchmark simulation output**

Function	Type of summary table created
`make_benchmark_aerosol_tables()`	Global aerosol burdens (1yr benchmarks only)
`make_benchmark_emis_tables()`	Emissions (by species & inventory)
`make_benchmark_mass_tables()`	Total mass of each species
`make_benchmark_mass_accumulation_tab`	Mass accumulation for each species
`make_benchmark_mass_conservation_tab`	Total mass of a single species at hourly intervals (to check mass conservation)
`make_benchmark_oh_metrics()`	Global OH metrics (mean OH, CH4 lifetime, methylchloroform lifetime)
`make_benchmark_operations_budget()`	Total mass of each species after each operation (transport, mixing, etc.)

The functions listed above create summary tables for quantities such as total mass of species, total mass of emissions, and OH metrics.

Many of these functions use pre-defined lists of variables in YAML files. If one dataset includes a variable but the other dataset does not, the data for that variable in the latter dataset will be considered to be NaN and will be plotted as such.

6.4.1 make_benchmark_aerosol_tables

This function creates tables of global aerosol budgets and burdens from GEOS-Chem 1-year full-chemistry benchmark simulation output.

```
def make_benchmark_aerosol_tables(
        devdir,
        devlist_aero,
        devlist_spc,
        devlist_met,
        devstr,
        year,
        days_per_mon,
        dst='./benchmark',
        overwrite=False,
        is_gchp=False,
        spcdb_dir=os.path.dirname(__file__)
):
    """
    Compute FullChemBenchmark aerosol budgets & burdens

    Args:
        devdir: str
            Path to development ("Dev") data directory
        devlist_aero: list of str
            List of Aerosols collection files (different months)
        devlist_spc: list of str
```

(continues on next page)

(continued from previous page)

```
            List of SpeciesConc collection files (different months)
        devlist_met: list of str
            List of meteorology collection files (different months)
        devstr: str
            Descriptive string for datasets (e.g. version number)
        year: str
            The year of the benchmark simulation (e.g. '2016').
        days_per_month: list of int
            List of number of days per month for all months

    Keyword Args (optional):
        dst: str
            Directory where budget tables will be created.
            Default value: './benchmark'
        overwrite: bool
            Overwrite burden & budget tables? (default=True)
            Default value: False
        is_gchp: bool
            Whether datasets are for GCHP
            Default value: False
        spcdb_dir: str
            Directory of species_datbase.yml file
            Default value: Directory of GCPy code repository

    """
```

6.4.2 make_benchmark_emis_tables

This function creates tables of emissions (by species and by inventory) from the output of GEOS-Chem benchmark simulations.

```
def make_benchmark_emis_tables(
        reflist,
        refstr,
        devlist,
        devstr,
        dst="./benchmark",
        benchmark_type="FullChemBenchmark",
        refmet=None,
        devmet=None,
        overwrite=False,
        ref_interval=[2678400.0],
        dev_interval=[2678400.0],
        spcdb_dir=os.path.dirname(__file__)
):
    """
    Creates a text file containing emission totals by species and
    category for benchmarking purposes.

    Args:
        reflist: list of str
```

(continues on next page)

(continued from previous page)

```
                List with the path names of the emissions file or files
                (multiple months) that will constitute the "Ref"
                (aka "Reference") data set.
        refstr: str
            A string to describe ref (e.g. version number)
        devlist: list of str
                List with the path names of the emissions file or files
                (multiple months) that will constitute the "Dev"
                (aka "Development") data set
        devstr: str
            A string to describe dev (e.g. version number)

    Keyword Args (optional):
        dst: str
            A string denoting the destination folder where the file
            containing emissions totals will be written.
            Default value: ./benchmark
        benchmark_type: str
            A string denoting the type of benchmark output to plot, options are
            FullChemBenchmark, TransportTracersBenchmark or CH4Benchmark.
            Default value: "FullChemBenchmark"
        refmet: str
            Path name for ref meteorology
            Default value: None
        devmet: str
            Path name for dev meteorology
            Default value: None
        overwrite: bool
            Set this flag to True to overwrite files in the
            destination folder (specified by the dst argument).
            Default value: False
        ref_interval: list of float
            The length of the ref data interval in seconds. By default, interval
            is set to [2678400.0], which is the number of seconds in July
            (our 1-month benchmarking month).
            Default value: [2678400.0]
        dev_interval: list of float
            The length of the dev data interval in seconds. By default, interval
            is set to [2678400.0], which is the number of seconds in July
            (our 1-month benchmarking month).
            Default value: [2678400.0]
        spcdb_dir: str
            Directory of species_datbase.yml file
            Default value: Directory of GCPy code repository

    """
```

6.4.3 make_benchmark_mass_tables

This function creates tables of total mass for species in two different GEOS-Chem benchmark simulations.

```
def make_benchmark_mass_tables(
        ref,
        refstr,
        dev,
        devstr,
        varlist=None,
        dst="./benchmark",
        subdst=None,
        overwrite=False,
        verbose=False,
        label="at end of simulation",
        spcdb_dir=os.path.dirname(__file__),
        ref_met_extra=None,
        dev_met_extra=None
):
    """
    Creates a text file containing global mass totals by species and
    category for benchmarking purposes.

    Args:
        reflist: str
            Pathname that will constitute
            the "Ref" (aka "Reference") data set.
        refstr: str
            A string to describe ref (e.g. version number)
        dev: list of str
            Pathname that will constitute
            the "Dev" (aka "Development") data set.  The "Dev"
            data set will be compared against the "Ref" data set.
        devstr: str
            A string to describe dev (e.g. version number)

    Keyword Args (optional):
        varlist: list of str
            List of variables to include in the list of totals.
            If omitted, then all variables that are found in either
            "Ref" or "Dev" will be included.  The varlist argument
            can be a useful way of reducing the number of
            variables during debugging and testing.
            Default value: None
        dst: str
            A string denoting the destination folder where the file
            containing emissions totals will be written.
            Default value: ./benchmark
        subdst: str
            A string denoting the sub-directory of dst where PDF
            files containing plots will be written.  In practice,
            subdst is only needed for the 1-year benchmark output,
            and denotes a date string (such as "Jan2016") that
```

(continues on next page)

(continued from previous page)

```
            corresponds to the month that is being plotted.
            Default value: None
        overwrite: bool
            Set this flag to True to overwrite files in the
            destination folder (specified by the dst argument).
            Default value: False
        verbose: bool
            Set this flag to True to print extra informational output.
            Default value: False.
        spcdb_dir: str
            Directory of species_datbase.yml file
            Default value: Directory of GCPy code repository
        ref_met_extra: str
            Path to ref Met file containing area data for use with restart files
            which do not contain the Area variable.
            Default value: ''
        dev_met_extra: str
            Path to dev Met file containing area data for use with restart files
            which do not contain the Area variable.
            Default value: ''
    """
```

6.4.4 make_benchmark_mass_accumulation_tables

This function creates tables of mass accumulation over time for species in two different GEOS-Chem benchmark simulations.

```python
def create_mass_accumulation_table(
        refdatastart,
        refdataend,
        refstr,
        refperiodstr,
        devdatastart,
        devdataend,
        devstr,
        devperiodstr,
        varlist,
        met_and_masks,
        label,
        trop_only=False,
        outfilename="GlobalMassAccum_TropStrat.txt",
        verbose=False,
        spcdb_dir=os.path.dirname(__file__)
):
    """
    Creates a table of global mass accumulation for a list of species in
    two data sets.  The data sets, which typically represent output from two
    different model versions, are usually contained in netCDF data files.

    Args:
        refdatastart: xarray Dataset
```

(continues on next page)

(continued from previous page)

```
            The first data set to be compared (aka "Reference").
        refdataend: xarray Dataset
            The first data set to be compared (aka "Reference").
        refstr: str
            A string that can be used to identify refdata
            (e.g. a model version number or other identifier).
        refperiodstr: str
            Ref simulation period start and end
        devdatastart: xarray Dataset
            The second data set to be compared (aka "Development").
        devdataend: xarray Dataset
            The second data set to be compared (aka "Development").
        devstr: str
            A string that can be used to identify the data set specified
            by devfile (e.g. a model version number or other identifier).
        devperiodstr: str
            Ref simulation period start and end
        varlist: list of strings
            List of species concentation variable names to include
            in the list of global totals.
        met_and_masks: dict of xarray DataArray
            Dictionary containing the meterological variables and
            masks for the Ref and Dev datasets.
        label: str
            Label to go in the header string.  Can be used to
            pass the month & year.

    Keyword Args (optional):
        trop_only: bool
            Set this switch to True if you wish to print totals
            only for the troposphere.
            Default value: False (i.e. print whole-atmosphere totals).
        outfilename: str
            Name of the text file which will contain the table of
            emissions totals.
            Default value: "GlobalMass_TropStrat.txt"
        verbose: bool
            Set this switch to True if you wish to print out extra
            informational messages.
            Default value: False
        spcdb_dir: str
            Directory of species_datbase.yml file
            Default value: Directory of GCPy code repository

    Remarks:
        This method is mainly intended for model benchmarking purposes,
        rather than as a general-purpose tool.

        Species properties (such as molecular weights) are read from a
        YAML file called "species_database.yml".
    """
```

6.4.5 make_benchmark_mass_conservation_table

This function creates a timeseries table of the global mass of the `PassiveTracer` species. Usually used with output from 1-year TransportTracers benchmark simulations.

```python
def make_benchmark_mass_conservation_table(
        datafiles,
        runstr,
        dst="./benchmark",
        overwrite=False,
        areapath=None,
        spcdb_dir=os.path.dirname(__file__)
):
    """
    Creates a text file containing global mass of the PassiveTracer
    from Transport Tracer simulations across a series of restart files.

    Args:
        datafiles: list of str
            Path names of restart files.
        runstr: str
            Name to put in the filename and header of the output file
        refstr: str
            A string to describe ref (e.g. version number)
        dev: str
            Path name of "Dev" (aka "Development") data set file.
            The "Dev" data set will be compared against the "Ref" data set.
        devmet: list of str
            Path name of dev meteorology data set.
        devstr: str
            A string to describe dev (e.g. version number)

    Keyword Args (optional):
        dst: str
            A string denoting the destination folder where the file
            containing emissions totals will be written.
            Default value: "./benchmark"
        overwrite: bool
            Set this flag to True to overwrite files in the
            destination folder (specified by the dst argument).
            Default value: False
        areapath: str
            Path to a restart file containing surface area data.
            Default value: None
        spcdb_dir: str
            Path to the species_database.yml
            Default value: points to gcpy/gcpy folder
    """
```

6.4.6 make_benchmark_oh_metrics

This function generates a table of OH metrics from GEOS-Chem benchmark simulation output.

```
def make_benchmark_oh_metrics(
        ref,
        refmet,
        refstr,
        dev,
        devmet,
        devstr,
        dst="./benchmark",
        overwrite=False,
):
    """
    Creates a text file containing metrics of global mean OH, MCF lifetime,
    and CH4 lifetime for benchmarking purposes.

    Args:
        ref: str
            Path name of "Ref" (aka "Reference") data set file.
        refmet: str
            Path name of ref meteorology data set.
        refstr: str
            A string to describe ref (e.g. version number)
        dev: str
            Path name of "Dev" (aka "Development") data set file.
            The "Dev" data set will be compared against the "Ref" data set.
        devmet: list of str
            Path name of dev meteorology data set.
        devstr: str
            A string to describe dev (e.g. version number)

    Keyword Args (optional):
        dst: str
            A string denoting the destination folder where the file
            containing emissions totals will be written.
            Default value: "./benchmark"
        overwrite: bool
            Set this flag to True to overwrite files in the
            destination folder (specified by the dst argument).
            Default value: False
    """
```

6.4.7 make_benchmark_operations_budget

Creates a table with the change in species mass after each GEOS-Chem operation, using output from GEOS-Chem benchmark simulations.

```
def make_benchmark_operations_budget(
        refstr,
        reffiles,
        devstr,
        devfiles,
        ref_interval,
        dev_interval,
        benchmark_type=None,
        label=None,
        col_sections=["Full", "Trop", "PBL", "Strat"],
        operations=[
             "Chemistry", "Convection", "EmisDryDep",
             "Mixing", "Transport", "WetDep"
        ],
        compute_accum=True,
        compute_restart=False,
        require_overlap=False,
        dst='.',
        species=None,
        overwrite=True,
        verbose=False,
        spcdb_dir=os.path.dirname(__file__)
):
    """
    Prints the "operations budget" (i.e. change in mass after
    each operation) from a GEOS-Chem benchmark simulation.

    Args:
        refstr: str
            Labels denoting the "Ref" versions
        reffiles: list of str
            Lists of files to read from the "Ref" version.
        devstr: str
            Labels denoting the "Dev" versions
        devfiles: list of str
            Lists of files to read from "Dev" version.
        interval: float
            Number of seconds in the diagnostic interval.

    Keyword Args (optional):
        benchmark_type: str
            A string denoting the type of benchmark output to plot, options are
            FullChemBenchmark, TransportTracersBenchmark, or CH4Benchmark.
            Default value: None
        label: str
            Contains the date or date range for each dataframe title.
            Default value: None
        col_sections: list of str
```

(continues on next page)

(continued from previous page)

```
            List of column sections to calculate global budgets for. May
            include Strat eventhough not calculated in GEOS-Chem, but Full
            and Trop must also be present to calculate Strat.
            Default value: ["Full", "Trop", "PBL", "Strat"]
        operations: list of str
            List of operations to calculate global budgets for. Accumulation
            should not be included. It will automatically be calculated if
            all GEOS-Chem budget operations are passed and optional arg
            compute_accum is True.
            Default value: ["Chemistry","Convection","EmisDryDep",
                            "Mixing","Transport","WetDep"]
        compute_accum: bool
            Optionally turn on/off accumulation calculation. If True, will
            only compute accumulation if all six GEOS-Chem operations budgets
            are computed. Otherwise a message will be printed warning that
            accumulation will not be calculated.
            Default value: True
        compute_accum: bool
            Optionally turn on/off accumulation calculation. If True, will
            only compute accumulation if all six GEOS-Chem operations budgets
            are computed. Otherwise a message will be printed warning that
            accumulation will not be calculated.
            Default value: True
        compute_restart: bool
            Optionally turn on/off calculation of mass change based on restart
            file. Only functional for "Full" column section.
            Default value: False
        require_overlap: bool
            Whether to calculate budgets for only species that are present in
            both Ref or Dev.
            Default value: False
        dst: str
            Directory where plots & tables will be created.
            Default value: '.' (directory in which function is called)
        species: list of str
            List of species for which budgets will be created.
            Default value: None (all species)
        overwrite: bool
            Denotes whether to overwrite existing budget file.
            Default value: True
        verbose: bool
            Set this switch to True if you wish to print out extra
            informational messages.
            Default value: False
    """
""
```

CHAPTER

SEVEN

SIX PANEL PLOTTING

This example script may also be found at gcpy/exampls/plotting/plot_comparisons.py.

```
#!/usr/bin/env python
"""
Six Panel Comparison Plots
--------------------------------------
This example script demonstrates the comparitive plotting
capabilities of GCPy, including single level plots as well as
global zonal mean plots. These comparison plots are frequently
used to evaluate results from different runs / versions of
GEOS-Chem, but can also be used to compare results from different
points in one run that are stored in separate xarray datasets.

The example data described here is in lat/lon format, but the same
code works equally well for cubed-sphere (GCHP) data.
"""
import xarray as xr
import matplotlib.pyplot as plt
from gcpy.constants import skip_these_vars
from gcpy.plot.compare_single_level import compare_single_level
from gcpy.plot.compare_zonal_mean import compare_zonal_mean


def main():
    """
    Example function to create six-panel comparison plots.
    """

    # xarray allows us to read in any NetCDF file, the format of
    # GEOS-Chem diagnostics, #as an xarray Dataset
    #
    # The skip_these_vars list avoids trying to read certain
    # GCHP variables that cause data read issues.
    ref_ds = xr.open_dataset(
        'first_run/GEOSChem.Restart.20160801_0000z.nc4',
        drop_variables=skip_these_vars
    )
    dev_ds = xr.open_dataset(
        'second_run/GEOSChem.Restart.20160801_0000z.nc4',
        drop_variables=skip_these_vars
```

(continues on next page)

(continued from previous page)

```
)

# ==================
# Single level plots
# ==================

# compare_single_level generates sets of six panel plots for
# data at a specified level in your datasets.  By default, the
# level at index 0 (likely the surface) is plotted. Here we will
# plot data at ~500 hPa, which is located at index 21 in the
# standard 72-level and 47-level GMAO vertical grids.
ilev=21

# You likely want to look at the same variables across both of
# your datasets. If a variable is in one dataset but not the other,
# the plots will show NaN values for the latter.  You can pass
# variable names in a list to these comparison plotting functions
# (otherwise all variables will plot).
varlist = ['SpeciesRst_O3', 'SpeciesRst_CO2']

# compare_single_level has many arguments which can be optionally
# specified. The first four arguments are required.  They specify
# your first xarray Dataset, the name of your first dataset,
# your second xarray Dataset, and the name of your second dataset.
# Here we will also pass a specific level and the names of the
# variables you want to plot.
compare_single_level(
    ref_ds,
    'Dataset 1',
    dev_ds,
    'Dataset 2',
    ilev=ilev,
    varlist=varlist
)
plt.show()

# Using plt.show(), you can view the plots interactively.
# You can also save out the plots to a PDF.
compare_single_level(
    ref_ds,
    'Dataset 1',
    dev_ds,
    'Dataset 2',
    ilev=ilev,
    varlist=varlist,
    pdfname='single_level.pdf'
)

# ==================
# Zonal Mean Plots
# ==================
```

(continues on next page)

(continued from previous page)

```
    # compare_zonal_mean generates sets of six panel plots containing
    # zonal mean data across your dataset.  compare_zonal_mean shares
    # many of the same arguments as compare_single_level.  You can
    # specify pressure ranges in hPa for zonal mean plotting (by
    # default every vertical level is plotted)
    compare_zonal_mean(
        ref_ds,
        'Dataset 1',
        dev_ds,
        'Dataset 2',
        pres_range=[0, 100],
        varlist=varlist,
        pdfname='zonal_mean.pdf'
    )


# Only execute when we run as a standalone script
if __name__ == '__main__':
    main()
```

This example script may also be found at gcpy/examples/plotting/plot_single_panel.py.

CHAPTER

EIGHT

SINGLE PANEL PLOTTING

```
#!/usr/bin/env python
"""
Global and Regional Single Panel Plots
--------------------------------------
This example script demonstrates the core single panel plotting
capabilities of GCPy, including global and regional single level plots
as well as global zonal mean plots.

The example data described here is in lat/lon format, but the same code
works equally well for cubed-sphere (GCHP) data.

For full documentation on the plotting capabilities of GCPy
(including full argument lists), please see the GCPy documentation
at https://gcpy.readthedocs.io.
"""
import xarray as xr
import matplotlib.pyplot as plt
from gcpy.plot.single_panel import single_panel


def main():
    """
    Example routine to create single panel plots.
    """

    # xarray allows us to read in any NetCDF file, the format of
    # GEOS-Chem diagnostics as an xarray Dataset
    dset = xr.open_dataset('GEOSChem.Restart.20160701_0000z.nc4')

    # You can easily view the variables available for plotting
    # using xarray.  Each of these variables has its own xarray
    # DataArray within the larger Dataset container.
    print(dset.data_vars)

    # Most variables have some sort of prefix; in this example all
    # variables are prefixed with 'SpeciesRst_'. We'll select the
    # DataArray for ozone.
    darr = dset.SpeciesRst_O3

    # Printing a DataArray gives a summary of the dimensions and attributes
```

(continues on next page)

(continued from previous page)

```
    # of the data.
    print(darr)

    # This Restart file has a time dimension of size 1, with 72 vertical levels,
    #46 latitude indicies, and 72 longitude indices.


    # ==================
    # Single-level Plots
    # ==================

    # gcpy.single_panel is the core plotting function of GCPy, able to
    # create a one panel zonal mean or single level plot.  Here we will
    # create a single level plot of ozone at ~500 hPa.  We must manually
    # index into the level that we want to plot (index 22 in the standard
    # 72-layer and 47-layer GMAO vertical grids).
    slice_500 = darr.isel(lev=22)

    # single_panel has many arguments which can be optionally specified.
    # The only argument you must always pass to a call to single_panel is
    # the DataArray that you want to plot. By default, the created plot
    # includes a colorbar with units read from the DataArray, an
    # automatic title (the data variable name in the DataArray), and
    # an extent equivalent to the full lat/lon extent of the DataArray
    single_panel(slice_500)
    plt.show()

    #You can specify a specific area of the globe you would like plotted
    # using the 'extent' argument, which uses the format [min_longitude,
    # max_longitude, min_latitude, max_latitude] with bounds
    # [-180, 180, -90, 90]
    single_panel(slice_500, extent=[50, -90, -10, 60])
    plt.show()

    # Other commonly used arguments include specifying a title and a
    # colormap (defaulting to a White-Green-Yellow-Red colormap)
    #You can find more colormaps at
    # https://matplotlib.org/tutorials/colors/colormaps.html
    single_panel(
        slice_500,
        title='500mb Ozone over the North Pacific',
        comap=plt.get_cmap("viridis"),
        log_color_scale=True,
        extent=[80, -90, -10, 60]
    )
    plt.show()

    # ===================
    # Zonal Mean Plotting
    # ===================

    # Use the plot_type argument to specify zonal_mean plotting
```

(continues on next page)

(continued from previous page)

```
    single_panel(
        darr,
        plot_type="zonal_mean"
    )
    plt.show()

    #You can specify pressure ranges in hPa for zonal mean plot
    # (by default every vertical level is plotted)
    single_panel(
        darr,
        pres_range=[0, 100],
        log_yaxis=True,
        log_color_scale=True
    )
    plt.show()



# Only execute when we run as a standalone script
if __name__ == '__main__':
    main()
```

CHAPTER

NINE

PLOT TIMESERIES

This example script may also be found at gcpy/examples/plotting/plot_single_panel.py.

```python
#!/usr/bin/env python
'''
Example of plotting timeseries data from GEOS-Chem and saving
the output to a PDF file.  You can modify this for your particular
diagnostic output.  This also contains a good overview of

This example script creates a PDF file with 2 pages.

   Page 1:
   -------
       O3 from the first model layer (from the "SpeciesConc"
       diagnostic collection is) plotted in blue.

       O3 at 10 meter height (from the "SpeciesConc_10m"
       diagnostic collection) is plotted in red.

   Page 2:
   -------
       HNO3 from the first model layer (from the SpeciesConc
       diagnostic collection is) plotted in blue.

       HNO3 at 10 meter height (from the SpeciesConc_10m
       diagnostic collection) is plotted in red.

You can of course modify this for your own particular applications.

Author:
-------
Bob Yantosca
yantosca@seas.harvard.edu
23 Aug 2019
'''

# Imports
import os
import warnings
import numpy as np
import matplotlib.dates as mdates
```

(continues on next page)

(continued from previous page)

```python
import matplotlib.ticker as mticker
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import xarray as xr
from gcpy import constants


# Tell matplotlib not to look for an X-window, as we are plotting to
# a file and not to the screen.  This will avoid some warning messages.
os.environ['QT_QPA_PLATFORM'] = 'offscreen'

# Suppress harmless run-time warnings (mostly about underflow in division)
warnings.filterwarnings('ignore', category=RuntimeWarning)
warnings.filterwarnings('ignore', category=UserWarning)


def find_files_in_dir(path, substrs):
    '''
    Returns a list of all files in a directory that match one or more
    substrings.

    Args:
    -----
        path : str
            Path to the directory in which to search for files.

        substrs : list of str
            List of substrings used in the search for files.

    Returns:
    --------
        file_list : list of str
            List of files in the directory (specified by path)
            that match all substrings (specified in substrs).
    '''

    # Initialize
    file_list = []

    # Walk through the given data directory.  Then for each file found,
    # add it to file_list if it matches text in search_list.
    for root, directory, files in os.walk(path):
        for f in files:
            for s in substrs:
                if s in f:
                    file_list.append(os.path.join(root, f))

    # Return an alphabetically sorted list of files
    file_list.sort()
    return file_list
```

(continues on next page)

(continued from previous page)

```
def find_value_index(seq, val):
    """
    Finds the index of a numpy array that is close to a value.

    Args:
    -----
        seq : numpy ndarray
            An array of numeric values.

        val : number
            The value to search for in seq.

    Returns:
    --------
        result : integer
            The index of seq that has a value closest to val.

    Remarks:
    --------
    This algorithm was found on this page:
    https://stackoverflow.com/questions/48900977/find-all-indexes-of-a-numpy-array-
↪closest-to-a-value
    """
    r = np.where(np.diff(np.sign(seq - val)) != 0)
    idx = r + (val - seq[r]) / (seq[r + np.ones_like(r)] - seq[r])
    idx = np.append(idx, np.where(seq == val))
    idx = np.sort(idx)
    result = np.round(idx)

    # NOTE: xarray needs integer values, so convert here!
    return int(result[0])


def read_geoschem_data(path, collections):
    """
    Returns an xarray Dataset containing timeseries data.

    Args:
    -----
        path : str
            Directory path where GEOS-Chem diagnostic output
            files may be found.

        collections: list of str
            List of GEOS-Chem collections.  Files for these
            collections will be read into the xarray Dataset.

    Returns:
    --------
        ds : xarray Dataset
            A Dataset object containing the GEOS-Chem diagnostic
            output corresponding to the collections that were
```

(continues on next page)

(continued from previous page)

```
            specified.
    '''

    # Get a list of variables that GCPy should not read.
    # These are mostly variables introduced into GCHP with the MAPL v1.0.0
    # update.  These variables contain either repeated or non-standard
    # dimensions that can cause problems in xarray when combining datasets.
    skip_vars = constants.skip_these_vars

    # Find all files in the given
    file_list = find_files_in_dir(path, collections)

    # Return a single xarray Dataset containing data from all files
    # NOTE: Need to add combine="nested" for xarray 0.15 and higher
    v = xr.__version__.split(".")
    if int(v[0]) == 0 and int(v[1]) >= 15:
        return xr.open_mfdataset(file_list,
                                 drop_variables=skip_vars,
                                 combine="nested",
                                 concat_dim=None)
    else:
        return xr.open_mfdataset(file_list,
                                 drop_variables=skip_vars)


def plot_timeseries_data(ds, site_coords):
    '''
    Plots a timseries of data at a given (lat,lon) location.

    Args:
    -----
        ds : xarray Dataset
            Dataset containing GEOS-Chem timeseries data.

        site_coords : tuple
            Contains the coordinate (lat, lon) of a site location
            at which the timeseries data will be plotted.
    '''

    # ----------------------------------------------------------------------
    # Get the GEOS-Chem data for O3 and HNO3 corresponding to the
    # location of the observational station.  We will save these into
    # xarray DataArray objects, which we'll need for plotting.
    #
    # YOU CAN EDIT THIS FOR YOUR OWN PARTICULAR APPLICATION!
    # ----------------------------------------------------------------------

    # Find the indices corresponding to the site lon and lat
    lat_idx = find_value_index(ds.lat.values, site_coords[0])
    lon_idx = find_value_index(ds.lon.values, site_coords[1])

    # Save O3 from the first level (~60m height) (ppb) into a DataArray
```

(continues on next page)

(continued from previous page)

```
    O3_L1 = ds['SpeciesConc_O3'].isel(lon=lon_idx, lat=lat_idx, lev=0)
    O3_L1 *= 1.0e9
    O3_L1.attrs['units'] = 'ppbv'

    # Save O3 @ 10m height into a DataArray
    O3_10m = ds['SpeciesConc10m_O3'].isel(lon=lon_idx, lat=lat_idx)
    O3_10m *= 1.0e9
    O3_10m.attrs['units'] = 'ppbv'

    # Save HNO3 from the first level (~60m height) into a DataArray
    HNO3_L1 = ds['SpeciesConc_HNO3'].isel(lon=lon_idx, lat=lat_idx, lev=0)
    HNO3_L1 *= 1.0e9
    HNO3_L1.attrs['units'] = 'ppbv'

    # Save HNO3 @ 10m height into a DataArray
    HNO3_10m = ds['SpeciesConc10m_HNO3'].isel(lon=lon_idx, lat=lat_idx)
    HNO3_10m *= 1.0e9
    HNO3_10m.attrs['units'] = 'ppbv'

    # ----------------------------------------------------------------------
    # Create a PDF file of the plots
    # ----------------------------------------------------------------------

    # Get min & max days of the plot span (for setting the X-axis range).
    # To better center the plot, add a cushion of 12 hours on either end.
    time = ds['time'].values
    datemin = np.datetime64(time[0]) - np.timedelta64(12, 'h')
    datemax = np.datetime64(time[-1]) + np.timedelta64(12, 'h')

    # Define a PDF object so that we can save the plots to PDF
    pdf = PdfPages('O3_and_HNO3.pdf')

    # Loop over number of desired pages (in this case, 2)
    for i in range(0, 2):

        # Create a new figure: 1 plot per page, 2x as wide as high
        figs, ax0 = plt.subplots(1, 1, figsize=[12, 6])

        # -----------------------------
        # Plot O3 on the first page
        # -----------------------------
        if i == 0:

            # 1st model level
            O3_L1.plot.line(ax=ax0, x='time', color='blue',
                            marker='o', label='O3 from 1st model level',
                            linestyle='-')

            # 10 mheight
            O3_10m.plot.line(ax=ax0, x='time', color='red',
                             marker='x', label='O3 at 10m height',
                             linestyle='-')
```

(continues on next page)

(continued from previous page)

```
            # Set title (has to be after the line plots are drawn)
            ax0.set_title('O3 from the 1st model level and at 10m height')

            # Set Y-axis minor tick marks at every 2 ppb (5 intervals)
            ax0.yaxis.set_minor_locator(mticker.AutoMinorLocator(5))

            # Set y-axis title
            ax0.set_ylabel('O3 (ppbv)')

        # -----------------------------
        # Plot HNO3 on the second page
        # -----------------------------
        if i == 1:

            # 1st model level
            HNO3_L1.plot.line(ax=ax0, x='time', color='blue',
                              marker='o', label='HNO3 from 1st model level',
                              linestyle='-')

            # 10m height
            HNO3_10m.plot.line(ax=ax0, x='time', color='red',
                               marker='x', label='HNO3 at 10m height',
                               linestyle='-')

            # Set title (has to be after the line plots are drawn
            ax0.set_title('HNO3 from the 1st model level and at 10m height')

            # Set Y-axis minor tick marks at every 0.05 ppb (4 intervals)
            ax0.yaxis.set_minor_locator(mticker.AutoMinorLocator(4))

            # Set y-axis title
            ax0.set_ylabel('HNO3 (ppbv)')

        # -----------------------------
        # Set general plot parameters
        # -----------------------------

        # Add the plot legend
        ax0.legend()

        # Set the X-axis range
        ax0.set_xlim(datemin, datemax)

        # Set the X-axis major tickmarks
        locator = mdates.DayLocator()
        formatter = mdates.DateFormatter('%d')
        ax0.xaxis.set_major_locator(locator)
        ax0.xaxis.set_major_formatter(formatter)

        # Set X-axis minor tick marks at noon of each day
        # (i.e. split up the major interval into 2 bins)
```

(continues on next page)

(continued from previous page)

```
        ax0.xaxis.set_minor_locator(mticker.AutoMinorLocator(2))

        # Don't rotate the X-axis jtick labels
        ax0.xaxis.set_tick_params(rotation=0)

        # Center the X-axis tick labels
        for tick in ax0.xaxis.get_major_ticks():
            tick.label1.set_horizontalalignment('center')

        # Set X-axis and Y-axis labels
        ax0.set_xlabel('Day of July (and August) 2016')

        # -----------------------------
        # Save this page to PDF
        # -----------------------------
        pdf.savefig(figs)
        plt.close(figs)

    # ---------------------------------------------------------------------
    # Save the PDF file to disk
    # ---------------------------------------------------------------------
    pdf.close()


def main():
    """
    Main program.
    """
    # Path where the data files live
    # (YOU MUST EDIT THIS FOR YUR OWN PARTICULAR APPLICATION!)
    path_to_data = '/path/to/GEOS-Chem/diagnostic/data/files'

    # Get a list of files in the ConcAboveSfc and SpeciesConc collections
    # (YOU CAN EDIT THIS FOR YOUR OWN PARTICULAR APPLICATION!)
    collections = ['ConcAboveSfc', 'SpeciesConc']

    # Read GEOS-Chem data into an xarray Dataset
    ds = read_geoschem_data(path_to_data, collections)

    # Plot timeseries data at Centerville, AL (32.94N, 87.18W)
    # (YOU CAN EDIT THIS FOR YOUR OWN PARTICULAR APPLICATION!)
    site_coords = (32.94, -87.18)
    plot_timeseries_data(ds, site_coords)


if __name__ == "__main__":
    main()
```

CHAPTER

TEN

CONTRIBUTING GUIDELINES

Thank you for looking into contributing to GCPy! GEOS-Chem is a grass-roots model that relies on contributions from community members like you. Whether you're new to GEOS-Chem or a longtime user, you're a valued member of the community, and we want you to feel empowered to contribute.

10.1 We use GitHub and ReadTheDocs

We use GitHub to host the GCPy source code, to track issues, user questions, and feature requests, and to accept pull requests: https://github.com/geoschem/gcpy. Please help out as you can in response to issues and user questions.

GCPy documentation can be found at gcpy.readthedocs.io.

10.2 When should I submit updates?

Submit bug fixes right away, as these will be given the highest priority. Please see "Support Guidelines" for more information.

The practical aspects of submitting code updates are listed below.

10.3 How can I submit updates?

We use GitHub Flow, so all changes happen through pull requests. This workflow is described here.

As the author you are responsible for:

- Testing your changes
- Updating the user documentation (if applicable)
- Supporting issues and questions related to your changes

10.3.1 Process for submitting code updates

1. Create or log into your GitHub account.
2. Fork the GCPy repository into your Github account.
3. Clone your fork of the GCPy repositories to your computer system.
4. Add your modifications into a new branch off the **main** branch.
5. Test your update thoroughly and make sure that it works.
6. Review the coding conventions and checklists for code and data updates listed below.
7. Create a pull request in GitHub.
8. The GEOS-Chem Support Team will add your updates into the development branch for an upcoming GCPy version.
9. If the benchmark simulations reveal a problem with your update, the GCST will request that you take further corrective action.

10.3.2 Coding conventions

GCPy includes contributions from many people and multiple organizations. Therefore, some inconsistent conventions are inevitable, but we ask that you do your best to be consistent with nearby code.

10.3.3 Checklist for submitting code updates

1. Include thorough comments in all submitted code.
2. Include full citations for references at the top of relevant source code modules.
3. Remove extraneous code updates (e.g. testing options, other science).

10.4 How can I request a new feature?

We accept feature requests through issues on GitHub. To request a new feature, **open a new issue** and select the feature request template. Please include all the information that migth be relevant, including the motivation for the feature.

10.5 How can I report a bug?

Please see **Support Guidelines**.

10.6 Where can I ask for help?

Please see **Support Guidelines**

CHAPTER

ELEVEN

SUPPORT GUIDELINES

GCPy support is maintained by the **GEOS-Chem Support Team (GCST)**, which is based jointly at Harvard University and Washington University in St. Louis.

We track bugs, user questions, and feature requests through **GitHub issues**. Please help out as you can in response to issues and user questions.

11.1 How to report a bug

We use GitHub to track issues. To report a bug, **open a new issue**. Please include your name, institution, and all relevant information, such as simulation log files and instructions for replicating the bug.

11.2 Where can I ask for help?

We use GitHub issues to support user questions. To ask a question, **open a new issue** and select the question template. Please include your name and institution in the issue.

11.3 What type of support can I expect?

We will be happy to assist you in resolving bugs and technical issues that arise when using GCPy. User support and outreach is an important part of our mission to support the International GEOS-Chem User Community.

Even though we can assist in several ways, we cannot possibly do everything. We rely on users being resourceful and willing to try to resolve problems on their own to the greatest extent possible.

If you have a science question rather than a technical question, you should contact the relevant GEOS-Chem Working Group(s) directly. But if you do not know whom to ask, you may open a new issue (See "Where can I ask for help" above) and we will be happy to direct your question to the appropriate person(s).

11.4 How to submit changes

Please see **Contributing Guidelines**.

11.5 How to request an enhancement

Please see **Contributing Guidelines**.

CHAPTER

TWELVE

EDITING THESE DOCS

This documentation is generated with Sphinx. This page describes how to contribute to the GCPy documentation.

12.1 Quick start

You need the Sphinx Python to build (and therefore edit) this documentation. Assuming you already have Python installed, install Sphinx:

```
$ pip install sphinx
```

To build the documentation, navigate to `gcpy/docs` and make the html target:

```
gcuser:~$ cd gcpy/docs
gcuser:~/gcpy/docs$ make html
```

This will generate the HTML documentation in `gcpy/docs/build/html` from the reST files in `gcpy/docs/source`. You can view this local HTML documentation by opening `index.html` in your web-browser.

Note: You can clean the documentation with `make clean`.

12.2 Learning reST

Writing reST can be a bit tricky at first. Whitespace matters (just like in Python), and some directives can be easily miswritten. Two important things you should know right away are:

- Indents are 3-spaces
- "Things" are separated by 1 blank line (e.g., a list or code-block following a paragraph should be separated from the paragraph by 1 blank line)

You should keep these in mind when you're first getting started. Dedicating an hour to learning reST will save you time in the long-run. Below are some good resources for learning reST.

- reStructuredText primer: (single best resource; however, it's better read than skimmed)
- Official reStructuredText reference (there is *a lot* of information here)
- Presentation by Eric Holscher (co-founder of Read The Docs) at DjangoCon US 2015 (the entire presentation is good, but reST is described from 9:03 to 21:04)
- YouTube tutorial by Audrey Tavares's

A good starting point would be Eric Holscher's presentations followed by reading the reStructuredText primer.

12.3 Style guidelines

Important: This documentation is written in semantic markup. This is important so that the documentation remains maintainable by the GEOS-Chem Support Team. Before contributing to this documentation, please review our style guidelines. When editing the documentation, please refrain from using elements with the wrong semantic meaning for aesthetic reasons. Aesthetic issues should be addressed by changes to the theme (not changes to reST files).

For **titles and headers**:

- H1 titles should be underlined by `#` characters
- H2 headers should be underlined by `-` characters
- H3 headers should be underlined by `^` characters
- H4 headers should be avoided, but if necessary, they should be underlined by `"` characters

File paths occuring in the text should use the `:literal:` role.

Inline code, or references to variables in code, occuring in the text should use the `:code:` role.

Code snippets should use `.. code-block:: <language>` directive like so

```
.. code-block:: python

   import gcpy
   print("hello world")
```

The language can be "none" to omit syntax highlighting.

For command line instructions, the "console" language should be used. The `$` should be used to denote the console's prompt. If the current working directory is relevant to the instructions, a prompt like `gcuser:~/path1/path2$` should be used.

Inline literals (such as the `$` above) should use the `:literal:` role.

CHAPTER THIRTEEN

RELEASING NEW VERSIONS

This page describes some of the steps required for releasing new versions of GCPy on Github, PyPi, and conda-forge.

1. For clarity, update version numbers to the new release in the following locations:
   - `setup.py`
   - `gcpy/_version.py`
   - `docs/source/conf.py`
   - `gcpy/benchmark/run_benchmark.py`
   - `gcpy/benchmark/modules/run_1yr_fullchem_benchmark.py`
   - `gcpy/benchmark/modules/run_1yr_tt_benchmark.py`
2. Update `CHANGELOG.md`
3. Merge **dev** into **main**
4. Publish the release on Github.
5. Install `twine` using `pip install twine` (if you haven't done this before).
6. To package GCPy for publication to PyPi, run the following from the root of your local GCPy repository:

   ```
   $ conda activate gcpy_env   # or whatever your conda env is named
   $ python setup.py sdist bdist_wheel
   $ twine check dist/*
   $ run twine upload --repository-url https://test.pypi.org/legacy/ dist/*
   ```

   Enter your login credentials for `test.pypi.org` as requested. Publishing to test.pypi ensures there are no issues with packaging the new release before publication to the primary PyPi database.
7. Publish to PyPi by running `run twine upload dist/*`, and enter your login information for pypi.org as requested.
8. Verify the new release is visible at https://pypi.org/project/geoschem-gcpy/ (may take a few minutes).
9. After a period of time (around an hour), you will be notified of a new PR at https://github.com/conda-forge/geoschem-gcpy-feedstock indicating conda-forge has detected a new release on PyPi. You should be able to merge this PR without any additinal interference once all checks have passed.
10. Once the feedstock PR has been merged and after another period of waiting, you should see builds for the new release when running `conda search -f geoschem-gcpy`. This indicates the new version is publicly available for installation through conda-forge.

INDEX