# GCPy

***Release 1.3.1***

**GEOS-Chem Support Team**

**Oct 26, 2022**

# 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 and xarray to simplify the task of working with 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 output.
3. Obtain GEOS-Chem's horizontal/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.

## 1.2 What GCPY was not intended to do

1. General NetCDF file modification: (crop a domain, extract some variables):
   - Use xarray instead.
   - Also see Work with netCDF data files at the GEOS-Chem ReadTheDocs site.
2. Statistical analysis:
   - Use scipy and scikit-learn tools instead.
3. Machine Learning:
   - Use the standard machine learning utilities (pytorch, tensorflow, julia, etc.).

CHAPTER

TWO

# INSTALLING GCPY

GCPy and its dependencies can be installed using Conda in either standard user mode (allow Conda to handle installation without Git support) or development mode using Conda and Conda-build (install from a Git clone). You can also manually install GCPy using a clone of the source code, but this option requires you to add the package to your `PYTHONPATH` manually and to install properly versioned dependencies on your own.

## 2.1 Requirements for GCPy

### 2.1.1 Software Prerequisites

GCPy is currently supported for Linux and MacOS operating systems. Due to a reliance on several packages without Windows support, **GCPy is not currently supported for Windows**. You will receive an error message if you attempt to install GCPy through Conda on Windows.

The only essential software package you need before installing GCPy is a distribution of the Conda package manager, which is used to install GCPy and its dependencies. It is also highly recommended that you create an environment in Conda for working with GCPy. Steps to setup Conda are described below:

1. Install Miniconda or Anaconda. Miniconda is much more lightweight and functions perfectly well for GCPy purposes, while Anaconda automatically includes many extra packages that are not directly relevant to GCPy.
2. After installing Miniconda or Anaconda, create a Conda environment for using GCPy. The basic usage (also found on the Conda Github hompeage) is:

```
# Navigate to the top-level GCPy folder
cd /path/to/gcpy

# Create a Conda environment for working with GCPy
conda env create -n gcpy_env --file=environment.yml

# Activate (enter) your new Conda environment
$ conda activate gcpy_env

# Deactivate (exit) your Conda environment
$ conda deactivate
```

   From within your Conda environment, you can follow the instructions on *Installing normally through Conda (if you don't plan on modifying GCPy source code)* or *Installing in development mode through Conda-build (for developers)*.

### 2.1.2 Python dependencies

Conda handles the installation of all dependencies for GCPy automatically. Most dependencies have minimum version requirements. We recommend using GCPy with Python 3.9. The list of dependencies (not including sub-dependencies) that are installed by Conda includes:

- Python 3.9
- cartopy
- matplotlib
- numpy
- scipy
- xarray
- xesmf
- esmpy
- pypdf2
- joblib
- xbpch
- pandas
- sparselt >= 0.1.3

A full list of package version requirements may be found in `docs/source/environment.yml`. There is also a symbolic link to this file from the top-level gcpy folder.

## 2.2 Installing GCPy for non-developers using Conda

GCPy is available through the `conda-forge` channel under the name `geoschem-gcpy`. Installing GCPy in your Conda environment requires two commands:

```
$ conda config --add channels conda-forge
$ conda install geoschem-gcpy
```

Conda will handle the installation of all dependencies and sub-dependencies for GCPy, which includes many Python packages and several non-Python libraries.

## 2.3 Installing GCPy for developers

If you wish to make changes to the GCPy source code with the goal of contributing to GCPy development, you will need to install GCPy from a clone of the GCPy Git repository:

```
$ git clone https://github.com/geoschem/gcpy.git
$ cd gcpy
$ conda config --add channels conda-forge
$ conda install geoschem-gcpy --only-deps
$ pip install -e .
```

Conda will handle the installation of dependencies when you install from this clone, and pip will point all GCPy links to this directory.

CHAPTER

THREE

# OVERVIEW OF CAPABILITIES

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

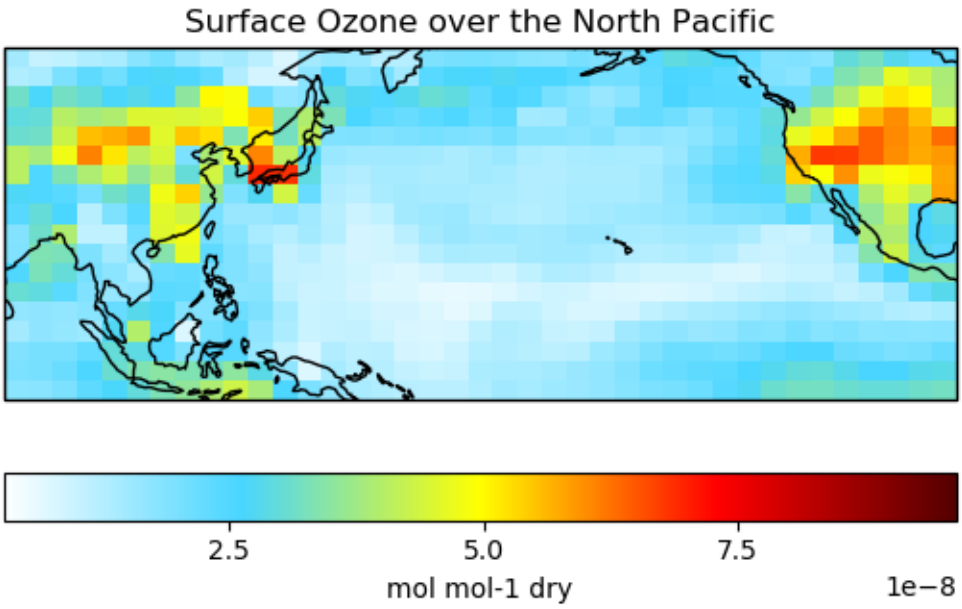
## 3.1 Spatial Plotting

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.

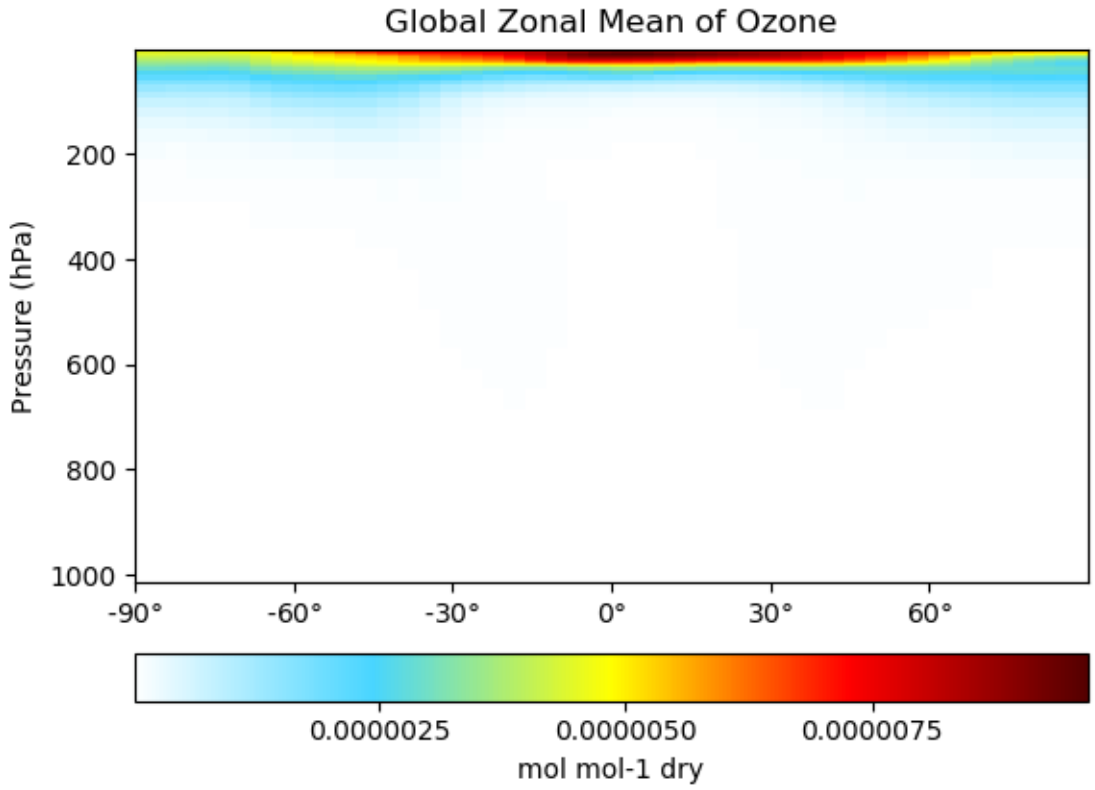
### 3.1.1 Single Panel Plots

Single panel plots are generated through the `plot.single_panel()` function. `plot.single_panel()` 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 gcpy.plot as gcplot
import matplotlib.pyplot as plt

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

# plot surface Ozone over the North Pacific
gcplot.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
gcplot.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

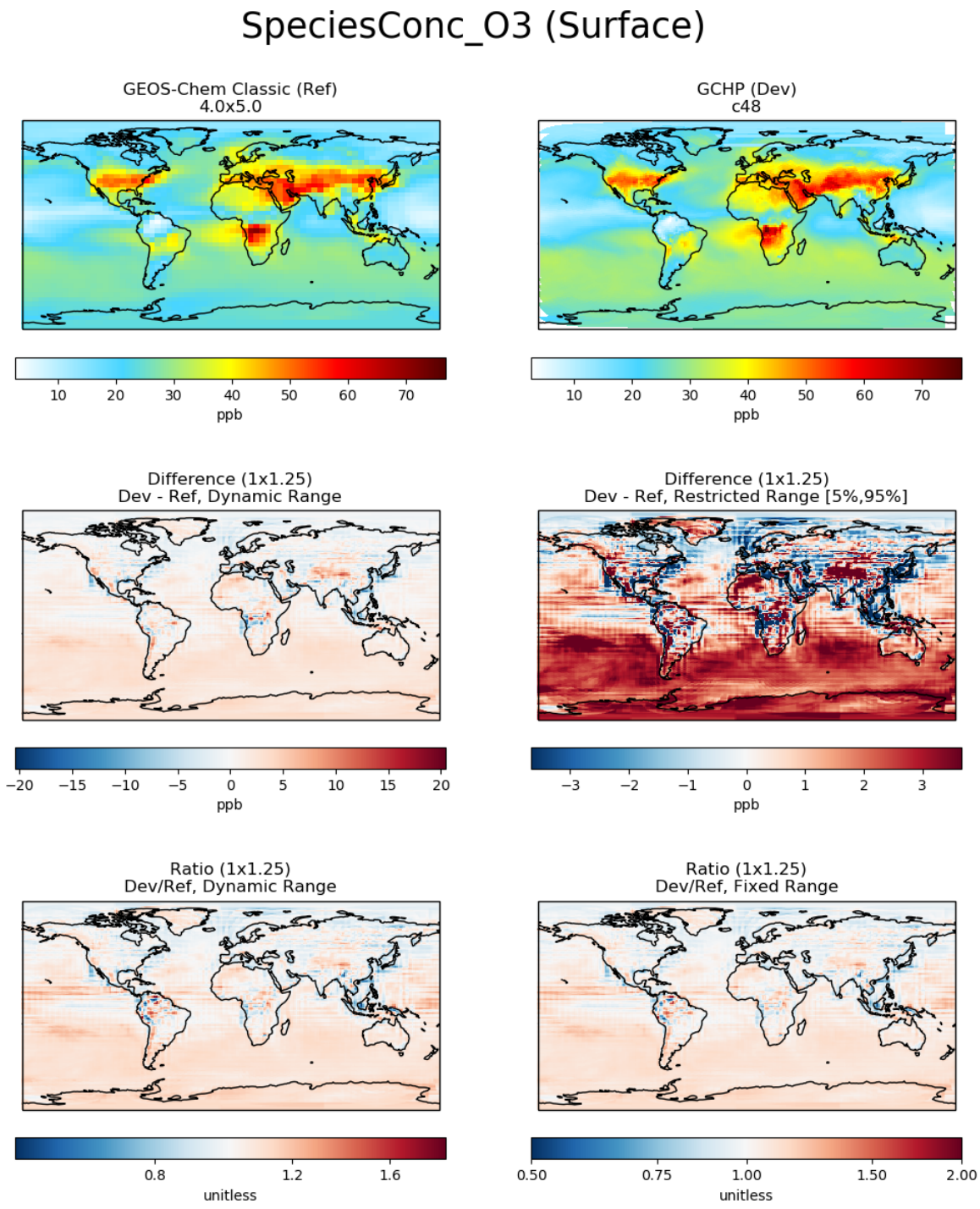
Six panel plots are used to compare results across two different model runs. Single level and zonal mean plotting options are both available. The two model runs do not need to be the same resolution or even the same grid type (GEOS-Chem Classic and GCHP output can be mixed at will).

```
import xarray as xr
import gcpy.plot as gcplot
import matplotlib.pyplot as plt

# 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
gcplot.compare_single_level(
    gcc_ds,
    'GEOS-Chem Classic',
```

(continues on next page)

(continued from previous page)

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

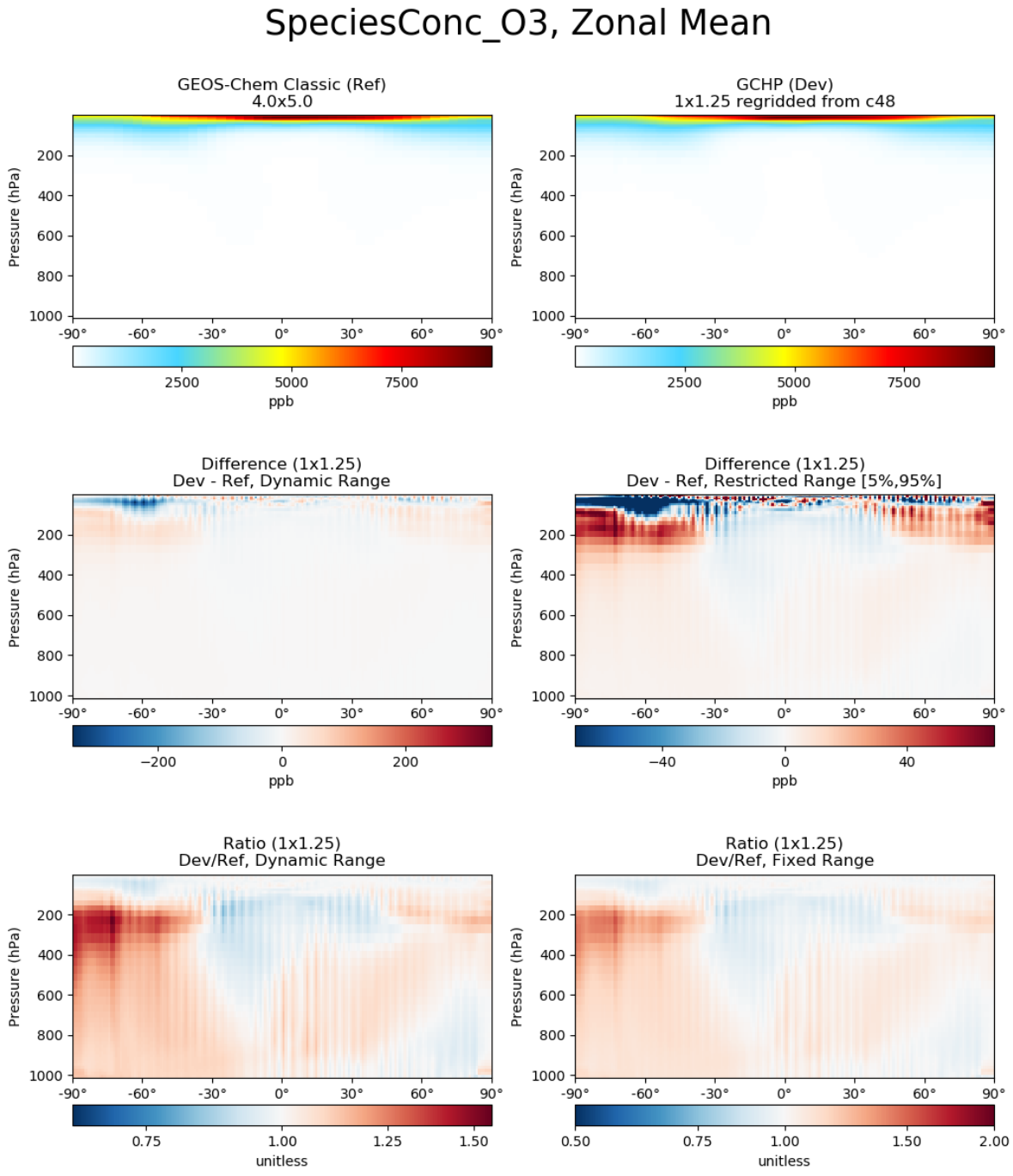



```
#Plot comparison of global zonal mean ozone
gcplot.compare_zonal_mean(
    gcc_ds,
    'GEOS-Chem Classic',
    gchp_ds,
```

(continues on next page)

(continued from previous page)

```
    '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 `benchmark.py` to generate full plots of benchmark diagnostics. Functions like `benchmark.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 GCST uses full benchmark plotting / table scripts similar to this example to produce plots and tables for official model benchmarks. Full documentation for the benchmark plotting functions can be found here.

## 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 `benchmark.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 |
+-------------+-------------+-------------+-----------+------------+
```

Full documentation for operations budget table creation can be found here.

### 3.2.2 Mass Tables

The `benchmark.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
```

Full documentation for mass table creation can be found here.

### 3.2.3 Emissions Tables

The `benchmark.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
```

Full documentation for emissions table creation can be found here.

## 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 NetCDF files between different grid types / resolutions and can be called from the command line or as a function.

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 plotting capabilities of GCPy, including possible argument values for every plotting function.

## 4.1 compare_single_level and compare_zonal_mean

`gcpy.plot.compare_single_level()` and `gcpy.plot.compare_zonal_mean()` both generate six panel plots comparing variables between two datasets. They share significant structural overlap both in output appearance and code implementation. This section gives an overview of the components of the plots generated by these functions, their shared arguments, and features unique to each function.

### 4.1.1 Shared structure

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.

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.

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

#### 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.

A basic script that calls `compare_zonal_mean()` or `compare_single_level()` looks like:

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

import xarray as xr
import gcpy.plot as gcplot
import matplotlib.pyplot as plt

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

:code`compare_single_level()` and `compare_zonal_mean()` also share many keyword arguments. Some of these arguments are plotting options that change the format of the plots, e.g. choosing to convert units to ug/m$^3$, which are automatically handled if you do not specify a value for that argument.

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.

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. On systems with higher (12+) core counts, the max 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.

### Shared keyword arguments:

**varlist** : list of str
: List of xarray dataset variable names to make plots for

  Default value: None (will compare all common variables)

**itime** : int
: Dataset time dimension index using 0-based system. Can only plot values from one time index in a single function call.

  Default value: 0

**refmet** : xarray.Dataset
: Dataset containing ref meteorology. Needed for area-based normalizations / ug/m3 unit conversions.

  Default value: None

**devmet** : xarray.Dataset
: Dataset containing dev meteorology. Needed for area-based normalizations and/or ug/m3 unit conversions.

Default value: None

**weightsdir** : str
Directory path for storing regridding weight files generated by xESMF.

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. The possible formats are 'int' (e.g. '48' for c48) for a cubed-sphere resolution or 'latxlon' (e.g. '4x5') for a lat/lon resolution.

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. This only applies to the top two panels of each plot.

Default value: True

**normalize_by_area** : bool
Set this flag to True to to normalize raw data in both Ref and Dev datasets by grid area. Either input ref and dev datasets must include AREA variable in m2 if normalizing by area, or refmet and devmet datasets must include Met_AREAM2 variable.

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** : bool
Whether to convert data units to ug/m3 for plotting. refmet and devmet cannot be None if converting to ug/m3.

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

**use_cmap_RdBu** : bool
Set this flag to True to use a blue-white-red colormap for plotting raw ref and dev data (the top two panels).

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 enable plotting data (only the top two panels, 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

**n_job** : int
Defines the number of simultaneous workers for parallel plotting. Only applicable when saving to PDF. 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: []

**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. This file is used for unit conversions to ug/m3. GEOS-Chem run directories include a copy of this file which may be more up-to-date than the version included with GCPy.

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)

### 4.1.2 compare_single_level

```
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 = [-1000, -1000, -1000, -1000],
    n_job=-1, sigdiff_list=[], 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
):
```

`compare_single_level()` features several keyword arguments that are not relevant to `compare_zonal_mean()`, including specifying which level to plot, the lat/lon extent of the plots, and which underlying `matplotlib.plot` function to use for plotting.

#### Function-specific keyword arguments:

**ilev** : int
Dataset level dimension index using 0-based system

Default value: 0

**extent** : list of float
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]

**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'

****extra_plot_args**
Any extra keyword arguments are passed through the plotting functions to be used in calls to `pcolormesh()` (CS) or `imshow()` (Lat/Lon).

### 4.1.3 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=[0, 2000],
    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=[],
    second_ref=None, second_dev=None, spcdb_dir=os.path.dirname(__file__),
    sg_ref_path='', sg_dev_path='', ref_vert_params=[[],[]],
    dev_vert_params=[[],[]], **extra_plot_args
):
```

`compare_zonal_mean()` features several keyword arguments that are not relevant to `compare_single_level()`, including specifying the pressure range to plot (defaulting to the complete atmosphere), whether the y-axis of the plots (pressure) should be in log format, and hybrid vertical grid parameters to pass if one or more of Ref and Dev do not use the typical 72-level or 47-level grids.

#### Function-specific keyword arguments:

**pres_range** : list of ints
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]

**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

**ref_vert_params** : list 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: [[], []]

**dev_vert_params** : list 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: [[], []]

****extra_plot_args**
Any extra keyword arguments are passed through the plotting functions to be used in calls to pcolormesh().

## 4.2 Single_panel

```
def single_panel(plot_vals, ax=None, plot_type="single_level",
    grid={}, gridtype="", title="fill",comap=WhGrYlRd,
    norm=[],unit="",extent=(None, None, None, None),
    masked_data=None,use_cmap_RdBu=False,
    log_color_scale=False, add_cb=True,
    pres_range=[0, 2000], pedge=np.full((1, 1), -1),
    pedge_ind=np.full((1,1), -1), log_yaxis=False,
    xtick_positions=[], xticklabels=[], proj=ccrs.PlateCarree(),
    sg_path='', ll_plot_func="imshow", vert_params=[[],[]],
    pdfname="", return_list_of_plots=False **extra_plot_args
):
```

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.

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

import xarray as xr
import gcpy.plot as gcplot
import matplotlib.pyplot as plt

ds = xr.open_dataset('GEOSChem.SpeciesConc.20160701_0000z.nc4')
#get surface ozone
plot_data = ds['SpeciesConc_O3'].isel(lev=0)

gcplot.single_panel(plot_data)
plt.show()
```

Currently single_panel() expects data with a 1-length ( or non-existent) time dimension, as well as a 1-length or non-existent Z dimension for single level plotting, so you'll need to do some pre-processing of your input data as shown in the above code snippet.

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.

### 4.2.1 Arguments:

In addition to the specific arguments listed below, any other keyword arguments will be forwarded to `matplotlib.pyplot.imshow()` / `matplotlib.pyplot.pcolormesh()`.

**plot_vals** : `xarray.DataArray or numpy array`
Single data variable GEOS-Chem output to plot

**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: [] (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, None, None, 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
Set this flag to True to add a colorbar to the plot

Default value: True

**pres_range** : list of 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 of float
Locations of lat/lon or lon ticks on plot

Default value: [] (will place automatically for zonal mean plots)

**xticklabels** : list of str
Labels for lat/lon ticks

Default value: [] (will determine automatically from xtick_positions)

**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: [[], []]

**pdfname** : str
File path to save plots as PDF

Default value: "" (will not create PDF)

**extra_plot_args** : various
Any extra keyword arguments are passed to calls to pcolormesh() (CS) or imshow() (Lat/Lon).

### 4.2.2 Function-specific return value:

`single_panel()` returns the following object:

**plot** : `matplotlib plot`
Plot object created from input

## 4.3 Benchmark Plotting Functions

`gcpy.benchmark` contains several functions for plotting GEOS-Chem output in formats requested by the GEOS-Chem Steering Committee. The primary use of these functions is to create plots of most GEOS-Chem output variables divided into specific categories, e.g. species categories such as Aerosols or Bromine for the SpeciesConc diagnostic. In each category, these functions create single level PDFs for the surface and 500hPa and zonal mean PDFs for the entire a tmosphere 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 these functions use pre-defined (via YAML files included in GCPy) lists of variables. 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.

### 4.3.1 Shared structure of benchmark functions

Each of the `gcpy.benchmark.make_benchmark_*_plots()` functions requires 4 arguments to specify the ref and dev datasets.

#### Shared arguments:

**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)

Note that the `ref` and `dev` arguments in `make_benchmark_*_plots()` are the paths to NetCDF files, rather than xarray Datasets as in `compare_single_level()` and `compare_zonal_mean()`. The `make_benchmark_*_plots()` functions internally open these files as xarray Datasets and pass those datasets to `compare_single_level()` and `compare_zonal_mean()`.

The benchmark plotting functions share several keyword arguments. Keyword arguments that do not share the same purpose across benchmark plotting functions have `NOTE:` in the description.

### Shared keyword arguments:

**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. NOTE: Not available in wetdep_plots

Default value: None

**overwrite** : bool
Set this flag to True to overwrite previously created 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 (the top two panels of each plot, not diffs) on a log color scale.

Default value: False

**sigdiff_files** : list of str
Filenames that will contain the list of quantities having significant differences between datasets. Three files are used: one for surface, one for 500hPa, and one for zonal mean. These lists are needed in order to fill out the benchmark approval forms.

---

**Note:** Not available in wetdep_plots

---

Default value: None

**spcdb_dir** : str
Directory containing species_database.yml file. This file is used for unit conversions to ug/m3. GEOS-Chem run directories include a copy of this file which may be more up-to-date than the version included with GCPy.

Default value: Path of GCPy code repository

**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.

---

**Note:** In `make_benchmark_conc_plots()`, parallelization occurs at the species category level. In all other functions, parallelization occurs within calls to `compare_single_level()` and `compare_zonal_mean()`.

Default value: -1 in `make_benchmark_conc_plots`, 1 in all others

---

### 4.3.2 make_benchmark_aod_plots

```
def make_benchmark_aod_plots(ref, refstr, dev, devstr, varlist=None,
   dst="./benchmark", subdst=None, overwrite=False, verbose=False,
   log_color_scale=False, sigdiff_files=None, weightsdir='.', n_job=-1,
   spcdb_dir=os.path.dirname(__file__)
):

"""
   Creates PDF files containing plots of column aerosol optical
   depths (AODs) for model benchmarking purposes.
"""
```

Function-specific keyword args:

**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

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

### 4.3.3 make_benchmark_conc_plots

```
def make_benchmark_conc_plots(ref, refstr, dev, devstr, dst="./benchmark",
   subdst=None, overwrite=False, verbose=False, collection="SpeciesConc",
   benchmark_type="FullChemBenchmark", 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
   second_dev=None, spcdb_dir=os.path.dirname(__file__)
):

"""
   Creates PDF files containing plots of species concentration
   for model benchmarking purposes.
"""
```

#### Function-specific keyword arguments:

**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, either FullChemBenchmark or TransportTracersBenchmark.

  Default value: “FullChemBenchmark”

**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

**restrict_cats** : list of str
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 str
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

**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

**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

This function creates species concentration plots using the `SpeciesConc` diagnostic output by default. 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) as follows:

```
"""
FullChemBenchmark:
   Aerosols:
      Dust: DST1, DST2, DST3, DST4
      Inorganic: NH4, NIT, SO4
      OC_BC: BCPI, BCPO, OCPI, OCPO
      SOA: Complex_SOA, Simple_SOA
      Sea_Salt: AERI, BrSALA, BrSALC, ISALA, ISALC, NITs,
```

(continues on next page)

(continued from previous page)

```
          SALA, SALAAL, SALACL, SALC, SALCAL, SALCCL, SO4s
   Bromine: Bry, BrOx, Br, Br2, BrCl, BrNO2, BrNO3, BrO,
      CH3Br, CH2Br2, CHBr3, HOBr, HBr
   Chlorine: Cly, ClOx, Cl, ClO, Cl2, Cl2O2, ClOO, ClNO2, ClNO3,
      CCl4, CFCs, CH3Cl, CH2Cl2, CH3CCl3, CHCl3, HOCl, HCl, Halons, HCFCs, OClO
   Iodine: Iy, IxOy, I, I2, IBr, ICl, IO, ION, IONO2, CH3I, CH2I2,
      CH2ICl, CH2IBr, HI, HOI, OIO
   Nitrogen: NOy, NOx, HNO2, HNO3, HNO4, MPAN, NIT, 'NO', NO2, NO3,
       N2O5, MPN, PAN, PPN, N2O, NHx, NH3, NH4, MENO3, ETNO3, IPRNO3, NPRNO3
   Oxidants: O3, CO, OH, NOx
   Primary_Organics:
      Alcohols: EOH, MOH
      Biogenics: ISOP, MTPA, MTPO, LIMO
      HCs: ALK4, BENZ, CH4, C2H6, C3H8, PRPE, TOLU, XYLE
      ROy: H2O2, H, H2, H2O, HO2, O1D, OH, RO2
   Secondary_Organic_Aerosols:
      Complex_SOA: TSOA0, TSOA1, TSOA2, TSOA3, ASOA1, ASOA2, ASOA3,
          ASOAN, TSOG0, TSOG1, TSOG2, TSOG3, ASOG1, ASOG2, ASOG3
      Isoprene_SOA: INDIOL, LVOCOA, SOAIE, SOAGX
      Simple_SOA: SOAP, SOAS
   Secondary_Organics:
      Acids: ACTA
      Aldehydes: ALD2, CH2O, HPALDs, MACR
      Epoxides: IEPOX
      Ketones: ACET, MEK, MVK
      Nitrates: ISOPN
      Other: GLYX, HCOOH, MAP, RCHO
      Peroxides: MP
   Sulfur: SOx, DMS, OCS, SO2, SO4
TransportTracersBenchmark:
   RnPbBeTracers: Rn222, Pb210, Pb210Strat, Be7, Be7Strat, Be10, Be10Strat
   PassiveTracers: PassiveTracer, SF6Tracer, CH3ITracer, COAnthroEmis25dayTracer,
       COAnthroEmis50dayTracer, COUniformEmis25dayTracer, GlobEmis90dayTracer,
       NHEmis90dayTracer, SHEmis90dayTracer

"""
```

make_benchmark_emis_plots

```
def make_benchmark_emis_plots(ref, refstr, dev, devstr, dst="./benchmark",
   subdst=None, plot_by_spc_cat=False, plot_by_hco_cat=False, overwrite=False,
   verbose=False,   flip_ref=False, flip_dev=False, log_color_scale=False,
   sigdiff_files=None, weightsdir='.', n_job=-1, 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.

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
```

(continues on next page)

(continued from previous page)

```
        column sums.
    """
```

#### Function-specific keyword args:

**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

**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

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

### 4.3.4 make_benchmark_jvalue_plots

```
def make_benchmark_jvalue_plots(ref, refstr, dev, devstr, varlist=None,
    dst="./benchmark", subdst=None, local_noon_jvalues=False,
    plots=["sfc", "500hpa", "zonalmean"],overwrite=False, verbose=False,
    flip_ref=False, flip_dev=False, log_color_scale=False, sigdiff_files=None,
    weightsdir='.', n_job=-1, spcdb_dir=os.path.dirname(__file__)
):
"""
   Creates PDF files containing plots of J-values for model
   benchmarking purposes.

   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
       plots up into separate files per category (e.g. Oxidants,
```

(continues on next page)

(continued from previous page)

```
       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.
"""
```

#### Function-specific keyword args:

**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

**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

**plots** : list of strings
: List of plot types to create.

  Default value: [‘sfc’, ‘500hpa’, ‘zonalmean’]

**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

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

### 4.3.5 make_benchmark_wetdep_plots

```
def make_benchmark_wetdep_plots(ref, refstr, dev, devstr, collection,
    dst="./benchmark", 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, spcdb_dir=os.path.dirname(__file__)
):
"""
   Creates PDF files containing plots of species concentration
   for model benchmarking purposes.
"""
```

### Function-specific keyword args:

**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, either FullChemBenchmark or TransportTracersBenchmark.

Default value: "FullChemBenchmark"

**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

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`:

```
"""
   WetLossConv: Pb210, Pb210Strat, Be7, Be7Strat, Be10, Be10Strat
   WetLossLS: Pb210, Pb210Strat, Be7, Be7Strat, Be10, Be10Strat
"""
```

CHAPTER

FIVE

# TABLING

This page describes the tabling capabilities of GCPy, including possible argument values for every tabling function. These functions are primarily used for model benchmarking purposes. All tables are printed to text files.

## 5.1 Emissions tables

```
def make_benchmark_emis_tables(reflist, refstr, devlist,
    devstr, dst="./benchmark", 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.
    """
```

### 5.1.1 Arguments:

**reflist:** list of str
: 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)

### 5.1.2 Keyword arguments:

**dst** : str
A string denoting the destination folder where the file containing emissions totals will be written.

Default value: ./benchmark

**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

`gcpy.benchmark.make_benchmark_emis_tables()` generates tables of total emissions categorized by species or by inventory. These tables contain total global emissions over the lengths of the Ref and Dev datasets, as well as the differences between totals across the two datasets. Passing a list of datasets as Ref or Dev (e.g. multiple months of emissions files) will result in printing totals emissions summed across all files in the list. Make sure to update the literal:*ref_interval* and/or `dev_interval` arguments if you pass input that does not correspond with 1 31 day month.

## 5.2 Mass Tables

```
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='', dev_met_extra=''
):
   """
   Creates a text file containing global mass totals by species and
   category for benchmarking purposes.
   """
```

### 5.2.1 Arguments:

**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)

### 5.2.2 Keyword arguments:

**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 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: ''

`gcpy.benchmark.make_benchmark_mass_tables` is used to create global mass tables of GEOS-Chem species from a `Restart` file. This function will create one table of total mass by species from the earth's surface to the top of the stratosphere and one table for only the troposphere. The tables contain total mass for each of the ref and dev datasets in Gg, as well as absolute and percentage difference between the two datasets. If your restart files do not contain an Area variable (`AREA` for GEOS-Chem Classic or `Met_AREAM2` for GCHP) then you will need to use the `ref_met_extra` and/or `dev_met_extra` arguments to pass the paths of NetCDF files containing the corresponding area variables (usually contained in meteorology diagnostic output).

## 5.3 Operations Budget Tables

```
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,
   require_overlap=False, dst='.', species=None, overwrite=True
):
   """
   Prints the "operations budget" (i.e. change in mass after
   each operation) from a GEOS-Chem benchmark simulation.
   """
```

### 5.3.1 Arguments:

**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.

### 5.3.2 Keyword arguments:

**benchmark_type** : str
: "TransportTracersBenchmark" or "FullChemBenchmark".

  Default value: None

**label** : str
: Contains the date or date range for each dataframe title.

  Default value: None

**col_sections** : list of str
: 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

**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

`gcpy.benchmark.make_benchmark_operations_budget()` creates tables of budgets for species separated by model operation. The tables show budgets for each of the ref and dev datasets in Gg, as well as absolute and percentage difference between the two datasets. Note that total accumulation across all operations will only be printed if you set `compute_accum==True` and all operations are included in `operations`. Note also that when using the non-local mixing scheme (default), `'Mixing'` includes emissions and dry deposition applied below the PBL. `'EmisDryDep'` therefore only captures fluxes above the PBL. When using full mixing, `'Mixing'` and `'EmisDryDep'` are fully separated.

## 5.4 Aerosol Budgets and Burdens

```
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
   """
```

### 5.4.1 Arguments:

**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`
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_mon :** `list of int`
List of number of days per month for all months

### 5.4.2 Keyword arguments:

**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

`gcpy.benchmark.make_benchmark_aerosol_tables()` generates two different tables using output from a single dataset. One contains annual mean aerosol burdens in Tg in the stratosphere, troposphere, and combined stratosphere and troposphere. The other table shows annual global mean AOD in the stratosphere, troposphere, and combined stratosphere and troposphere. Aerosol species used are pre-defined in `aod_species.yml`: BCPI, OCPI, SO4, DST1, SALA, and SALC.

CHAPTER

SIX

# REGRIDDING

This page describes the regridding capabilities of GCPy. GCPy currently supports regridding of data from GEOS-Chem restarts and output NetCDF files. 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. .. _regrid-plot:

## 6.1 Regridding for Plotting in GCPy

When plotting in GCPy (e.g. through `compare_single_level()` or `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.

### 6.1.1 Pass stretched-grid file paths

Stretched-grid parameters cannot currently be automatically determined from grid coordinates. If you are plotting stretched-grid data in `compare_single_level()` or `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.

### 6.1.2 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.

### 6.1.3 Automatic regridding decision process

When you do not specify a horizontal comparison resolution using the `cmpres` argument in `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.

## 6.2 Regridding Files

You can regrid existing GEOS-Chem restart or output diagnostic files between lat/lon and cubed-sphere formats using `gcpy.file_regrid`. `gcpy.file_regrid` can either be called directly from the command line using `python -m gcpy.file_regrid` or as a function (`gcpy.file_regrid.file_regrid()`) from a Python script or interpreter. The syntax of `file_regrid` is as follows:

```
def file_regrid(fin, fout, dim_format_in, dim_format_out,
cs_res_out=0, ll_res_out='0x0',
sg_params_in=[1.0, 170.0, -90.0], sg_params_out=[1.0, 170.0, -90.0]
):
"""
Regrids an input file to a new horizontal grid specification and saves it
as a new file.
"""
```

### 6.2.1 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 (choose from: classic, checkpoint, diagnostic), where classic denotes lat/lon and checkpoint / diagnostic are cubed-sphere formats

**dim_format_out** : str
: Format of the output file's dimensions (choose from: classic, checkpoint, diagnostic), where classic denotes lat/lon and checkpoint / diagnostic are cubed-sphere formats

### 6.2.2 Optional arguments:

**cs_res_out** : int
The cubed-sphere resolution of the output dataset. Not used if dim_format_out is classic.

Default value: 0

**ll_res_out** : str
The lat/lon resolution of the output dataset. Not used if dim_format_out is not classic/

Default value: '0x0'

**sg_params_in** : list[float, **float, float]**
Input grid stretching parameters [stretch-factor, target longitude, target latitude]. Not used if dim_format_in is classic

Default value: [1.0, 170.0, -90.0] (No stretching)

**sg_params_out** : list[float, **float, float]**
Output grid stretching parameters [stretch-factor, target longitude, target latitude]. Not used if dim_format_out is classic.

Default value: [1.0, 170.0, -90.0] (No stretching)

There are three dimension formats available for regridding: `classic` (GEOS-Chem Classic lat/lon format), `checkpoint` (GCHP restart file format), and `diagnostic` (GCHP output file format). You can regrid between any of these formats using `file_regrid`, as well as between different resolutions and/or grid-types within each dimension format (e.g. standard cubed-sphere checkpoint to stretched-grid checkpoint). Note that although the `cs_res_out` and `ll_res_out` parameters are technically optional in the function, you must specify at least one of these in your call to `file_regrid`.

As stated previously, you can either call `file_regrid.file_regrid()` directly or call it from the command line using `python -m gcpy.file_regrid ARGS`. An example command line call (separated by line for readability) for regridding a C90 cubed-sphere restart file to a C48 stretched-grid with a stretch factor of 3, a target longitude of 260.0, and a target latitude of 40.0 looks like:

```
python -m gcpy.file_regrid             \
      -i initial_GEOSChem_rst.c90_standard.nc   \
      --dim_format_in checkpoint      \
      -o sg_restart_c48_3_260_40.nc       \
      --cs_res_out 48            \
      --sg_params_out 3.0 260.0 40.0      \
      --dim_format_out checkpoint
```

## 6.3 Regridding with gridspec and sparselt

GCPy 1.3.0 and later supports regridding with the gridspec and sparselt utilities.

### 6.3.1 First-time setup

1. Install command line tool gridspec in your bin directory

```
$ pip install git+https://github.com/LiamBindle/gridspec.git
```

2. Make sure location of installation is added to path in your bashrc (or equivalent)

```
$ export PATH=/path/to/home/.local/bin:$PATH
```

3. Install sparselt as a python package.

```
$ conda install -c conda-forge sparselt==0.1.3
```

### 6.3.2 One-time setup per grid resolution combination

1. Create a directory structure to store files that you will use in regridding. Ideally this would be in a shared location where all of the GCPy users at your institution coud access it.

   Navigate to this directory.

```
$ mkdir /path/to/RegridInfo
```

2. Within this top level directory, create two directories that will store grid information and regridding weights. Navigate to the grid information folder.

```
$ mkdir Grids
$ mkdir Weights
$ cd Grids
```

3. Create tilefiles (if cubed-sphere) and grid spec file for each input and output grid resolution (see also gridspec README):

   For uniform cubed-sphere global grid, specify face side length.

   1. For simplicity, keep all cubed-sphere data in subdirectories of the Grids folder.

```
$ mkdir c24
$ gridspec-create gcs 24
$ mv c24*.nc c24

$ mkdir c48
$ gridspec-create gcs 48
$ mv c48*.nc c48

 ... etc for other grids ...
```

   2. For cubed-sphere stretched grid, specify face side length, stretch factor, and target latitude and longitude:

```
$ mkdir sc24
$ gridspec-create sgcs 24 -s 2 -t 40 -100
$ mv *c24*.nc sc24
```

   3. For uniform global lat-lon grid, specify the number of latitude and longitude grid boxes. For a list of optional settings, run the command **`gridspec-create latlon --help`**.

      Create a subdirectory named latlon and move all of your latlon grid specification files there.

```
$ gridspec-create latlon 90 180                 # Generic 1 x 1 grid
$ gridspec-create latlon 46 72 -dc -pc -hp      # GEOS-Chem Classic 4 x 5
$ gridspec-create latlon 91 144 -dc -pc -hp     # GEOS-Chem Classic 2 x 2.5
$ gridspec-create latlon 361 576 -dc -pc -hp    # MERRA-2 0.5 x 0.625
$ gridspec-create latlon 721 1172 -dc -pc -hp   # GEOS-FP 0.25 x  0.3125

$ mkdir latlon
$ mv regular_lat_lon*.nc latlon
```

4. (Optional) View contents of grid spec file:

```
$ gridspec-dump c24/c24_gridspec.nc

... etc. for other grids ...
```

5. Initialize your GCPy conda environmnt (which includes ESMF as a dependency):

```
$ conda activate gcpy_env
```

6. Navigate to the directory that will store the regridding weights. (Recall that we created this in created this in step #2.

```
$ cd /path/to/RegridInfo/Weights
```

7. Generate regridding weights (see also sparselt sample data files README), specifying the following:

   - Path to input file horizontal resolution grid spec netcdf file
   - Path to output file horizontal resolution grid spec netcdf file
   - Regridding type, e.g. conserve for conservative (string)
   - Name of output regridding weights file (include input and output resolutions)
   - Name of directory containing grid spec tilefiles

```
(gcpy_env) $ /ESMF_RegridWeightGen                                  \
             -s /path/to/RegridInfo/Grids/c48/c48_gridspec.nc       \
             -d /path/to/RegridInfo/Grids/regular_lat_lon_90x180.nc \
             -m conserve                                            \
             -w ./regrid_weights_c48_to_latlon90x180.nc             \
             --tilefile_path /path/to/RegridInfo/Grids/c48

... etc. for other grid combinations ...
```

8. (Optional) Consider using a bash script such as the one shown below if you need to create regridding weights to/from several grids.

```
#!/bin/bash

# Generates regridding weights with ESMF_RegridWeightGen

# The top-level directory containing Grids and Weights subdirectories
# (EDIT AS NEEDED)
main_dir="/path/to/RegridInfo"

# Subdirectories for grid specifications and regridding weights
grids_dir="${main_dir}/Grids"
weights_dir="${main_dir}/Weights"
```

(continues on next page)

(continued from previous page)

```
# GCHP cubed-sphere grids (EDIT AS NEEDED)
cs_list=(c24 c48 c90 c180 c360)

# GCClassic lat-lon grids (EDIT AS NEEDED)
ll_list=(46x72 91x144 361x576 721x1172)

# Loop over cubed-sphere grids
for cs in ${cs_list[@]}; do

    # Cubed-sphere gridspec file
    cs_grid_info="${grids_dir}/${cs}/${cs}_gridspec.nc"
    if [[ ! -f ${cs_grid_info} ]]; then
        echo "Could not find ${cs_grid_info}!"
        exit 1
    fi

    # Loop over latlon grids
    for ll in ${ll_list[@]}; do

        # Latlon gridspec file
        ll_grid_info="${grids_dir}/latlon/regular_lat_lon_${ll}.nc"
        if [[ ! -f ${ll_grid_info} ]]; then
            echo "Could not find ${ll_grid_info}!"
            exit 1
        fi

        # Cubed-sphere -> latlon regridding
        echo "----"
        echo "Regridding from ${cs} to ${ll}"
        weightfile="${weights_dir}/regrid_weights_${cs}_to_latlon${ll}.nc"
        ESMF_RegridWeightGen                    \
            -s ${cs_grid_info}                  \
            -d ${ll_grid_info}                  \
            -m conserve                         \
            -w ${weightfile}                    \
            --tilefile_path ${grids_dir}/${cs}
        unset weightfile

        # Latlon -> cubed-sphere regridding
        echo "----"
        echo "Regridding from ${ll} to ${cs}"
        weightfile="${weights_dir}/regrid_weights_latlon${ll}_to_${cs}.nc"
        ESMF_RegridWeightGen                    \
            -s ${ll_grid_info}                  \
            -d ${cs_grid_info}                  \
            -m conserve                         \
            -w ${weightfile}                    \
            --tilefile_path ${grids_dir}/${cs}
        unset weightfile

    done
done
```

### 6.3.3 Sample regridding script

Once you have created the tilefiles and regridding weights, you can use them to regrid data files. Shown below is a sample Python script that you can modify.

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

# Imports
import xarray as xr
import sparselt.esmf
import sparselt.xr

# Create a linear transform object from the regridding weights file
# for the combination of source and target horizontal resolutions.
transform = sparselt.esmf.load_weights(
    'path/to/RegridInfo/Weights/regrid_weights_c48_to_latlon90x180.nc',
     input_dims=[('nf', 'Ydim', 'Xdim'), (6, 48, 48)]
     output_dims=[('lat', 'lon'), (90, 180)],
)

# Open file to regrid as xarray DataSet.
ds = xr.open_dataset('my_data_c48.nc')

# Regrid the DataSet using the transform object.
ds = sparselt.xr.apply(transform, ds)

# Write xarray DataSet contents to netcdf file.
ds.to_netcdf("my_data_latlon90x180.nc")
```

CHAPTER

SEVEN

# SIX PANEL PLOTTING

```
#!/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.
"""

#xarray allows us to read in any NetCDF file, the format of most GEOS-Chem␣
↪diagnostics,
#as an xarray Dataset
import xarray as xr
ref_ds = xr.open_dataset('first_run/GEOSChem.Restart.20160801_0000z.nc4')
dev_ds = xr.open_dataset('second_run/GEOSChem.Restart.20160801_0000z.nc4')

import gcpy.plot as gcplot

"""
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']
```

(continues on next page)

(continued from previous page)

```
#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.
import matplotlib.pyplot as plt
gcplot.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.
gcplot.compare_single_level(ref_ds, 'Dataset 1', dev_ds, 'Dataset 2', ilev=ilev,␣
↪varlist=varlist, pdfname='single_level.pdf')

"""
Zonal Mean Plotting
-------------------
"""
#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)
gcplot.compare_zonal_mean(ref_ds, 'Dataset 1', dev_ds, 'Dataset 2', pres_range=[0,␣
↪100], varlist=varlist, pdfname='zonal_mean.pdf')
```

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.
"""

#xarray allows us to read in any NetCDF file, the format of most GEOS-Chem␣
↪diagnostics,
#as an xarray Dataset
import xarray as xr
ds = 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(ds.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.
da = ds.SpeciesRst_O3

#Printing a DataArray gives a summary of the dimensions and attributes of the data.
print(da)
#This Restart file has a time dimension of size 1, with 72 vertical levels,
#46 latitude indicies, and 72 longitude indices.
import gcpy.plot as gcplot

"""
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 = da.isel(lev=22)

#single_panel has many arguments which can be optionally specified. The only argument␣
↪you must always
```

(continues on next page)

(continued from previous page)

```
#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
import matplotlib.pyplot as plt
gcplot.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]
gcplot.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
gcplot.single_panel(slice_500, title='500mb Ozone over the North Pacific', comap =␣
↪plt.cm.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
gcplot.single_panel(da, plot_type="zonal_mean")
plt.show()

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

CHAPTER

NINE

# BENCHMARK PLOTTING / TABLING

Below is an example configuration file used to input the desired options for the comprehensive benchmark comparison script `run_benchmark.py`. Additional configuration file examples can be found in the `benchmarks` directory of GCpy.

The `run_benchmark.py` script allows one to perform benchmark comparisons between any simulation duration supplied in the configuration file provided the ref and dev simulations time periods match. Additionally, if the durations specified are exactly one year, then the corresponding `bmk_type` specialty comparison script will be run (either `run_1yr_fullchem_benchmark.py` or `run_1yr_tt_benchmark.py`). Any other duration will run the standard suite of benchmark comparisons.

To generate plots from a 1-month benchmark simulation, you would call `run_benchmark.py` as follows:

```
(gcpy_env) $ run_benchmark.py 1mo_benchmark.yml
```

Where `1mo_benchmark.yml` contains the following inputs:

```
&---
# ======================================================================
# Benchmark configuration file (**EDIT AS NEEDED**)
# customize in the following manner:
# (1) Edit the path variables so that they point to folders w/ model data
# (2) Edit the version strings for each benchmark simulation
# (3) Edit the switches that turn on/off creating of plots and tables
# (4) If necessary, edit labels for the dev and ref versions
# Note: When doing GCHP vs GCC comparisions gchp_dev will be compared
# to gcc_dev (not gcc_ref!). This ensures consistency in version names
# when doing GCHP vs GCC diff-of-diffs (mps, 6/27/19)
# ======================================================================
#
# Configuration for 1 month 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.
#
paths:
  main_dir: /n/holyscratch01/external_repos/GEOS-CHEM/gcgrid/geos-chem/validation/
↪gcpy_test_data/1mon
  results_dir: /path/to/BenchmarkResults
  weights_dir: /n/holyscratch01/external_repos/GEOS-CHEM/gcgrid/gcdata/ExtData/GCHP/
↪RegriddingWeights
```

(continues on next page)

(continued from previous page)

```
  spcdb_dir: default
#
# 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: "2019-07-01T00:00:00"
      bmk_end: "2019-08-01T00:00:00"
    gchp:
      version: GCHP_ref
      dir: GCHP_ref
      outputs_subdir: OutputDir
      restarts_subdir: Restarts
      bmk_start: "2019-07-01T00:00:00"
      bmk_end: "2019-08-01T00:00:00"
      is_pre_13.1: False
      is_pre_14.0: False
      resolution: c24
  dev:
    gcc:
      version: GCC_dev
      dir: GCC_dev
      outputs_subdir: OutputDir
      restarts_subdir: Restarts
      bmk_start: "2019-07-01T00:00:00"
      bmk_end: "2019-08-01T00:00:00"
    gchp:
      version: GCHP_dev
      dir: GCHP_dev
      outputs_subdir: OutputDir
      restarts_subdir: Restarts
      bmk_start: "2019-07-01T00:00:00"
      bmk_end: "2019-08-01T00:00:00"
      is_pre_13.1: False
      is_pre_14.0: False
      resolution: c24
#
# options: Specify the types of comparisons to perform
#
options:
  bmk_type: FullChemBenchmark
  gcpy_test: True # Specify if this is a gcpy test validation run
  comparisons:
    gcc_vs_gcc:
      run: True # True to run this comparison
```

(continues on next page)

(continued from previous page)

```
      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
#
# outputs: Types of output to generate (plots/tables)
#
  outputs:
    plot_conc: True
    plot_emis: True
    emis_table: True
    plot_jvalues: True
    plot_aod: True
    mass_table: True
    ops_budget_table: False
    OH_metrics: True
    ste_table: True # GCC only
    plot_options: # Plot concentrations and emissions by category?
      by_spc_cat: True
      by_hco_cat: True
```

YAML configuration files for 1-year benchmarks (`1yr_fullchem_benchmark.yml`, `1yr_tt_benchmark.yml`) are also provided in the `benchmarks` folder.

CHAPTER

TEN

# PLOT TIMESERIES

```
#!/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 gcpy.constants as gcon
import os
import numpy as np
import matplotlib.dates as mdates
import matplotlib.ticker as mticker
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import xarray as xr
import warnings
```

(continues on next page)

(continued from previous page)

```
# 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


   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.
```

(continues on next page)

(continued from previous page)

```
    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
        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 = gcon.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,
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
    # 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='-')

        # 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')
```

(continues on next page)

(continued from previous page)

```
        # 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)
      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'
```

(continues on next page)

(continued from previous page)

```
    # 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

# ELEVEN

# CONVERT BPCH TO NETCDF

```
#!/usr/bin/env python
'''
Example script that illustrates how to create a netCDF file
from an old GEOS-Chem binary punch ("bpch") file.
'''

# Imports
import gcpy
import xarray as xr
import xbpch as xb
import warnings

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

# ----------------------------------------------------------------------
# User configurable settings (EDIT THESE ACCORDINGLY)
# ----------------------------------------------------------------------

# Name of Bpch file
bpchfile = '/path/to/bpch/file'

# tracerinfo.dat and diaginfo,dat fiels
tinfo_file = '/path/to/tracerinfo.dat'
dinfo_file = '/path/to/diaginfo.dat'

# Name of netCDF file
ncfile = '/path/to/netcdf/file'

# Date string for the time:units attribute
datestr = 'YYYY-MM-DD'

# Number of seconds in the diagnostic interval (assume 1-month)
interval = 86400.0 * 31.0

# ----------------------------------------------------------------------
# Open the bpch file and save it into an xarray Dataset object
# NOTE: For best results, also specify the corresponding
# tracerinfo.dat diaginfo.dat metadata files.
# ----------------------------------------------------------------------
try:
   ds = xb.open_bpchdataset(filename=bpchfile,
                        tracerinfo_file=tinfo_file,
```

(continues on next page)

(continued from previous page)

```
                        diaginfo_file=dinfo_file)
except FileNotFoundError:
   print('Could not find file {}'.format(bpchfile))
   raise

# ----------------------------------------------------------------------
# Further manipulate the Dataset
# ----------------------------------------------------------------------

# Transpose the order of the xarray Dataset object read by
# xbpch so that its dimensions will be in the same order as
# Dataset objects read from netCDF files.
ds = ds.transpose()

# Convert the bpch variable names to the same naming
# convention as the netCDF ("History") diagnostics.
ds = gcpy.convert_bpch_names_to_netcdf_names(ds)

# xbpch does not include a time dimension, so we'll add one here
coords = ds.coords
coords['time'] = 0.0

# ----------------------------------------------------------------------
# Further edit variable attributes
# ----------------------------------------------------------------------
for v in ds.data_vars.keys():

   # Append time to the data array
   ds[v] = xr.concat([ds[v]], 'time')

   # Add long_name attribute for COARDS netCDF compliance
   ds[v].attrs['long_name'] = ds[v].attrs['full_name']

   # Remove some extraneous attributes that xbpch sets
   del ds[v].attrs['name']
   del ds[v].attrs['full_name']
   del ds[v].attrs['scale_factor']
   del ds[v].attrs['hydrocarbon']
   del ds[v].attrs['tracer']
   del ds[v].attrs['category']
   del ds[v].attrs['chemical']
   del ds[v].attrs['original_shape']
   del ds[v].attrs['origin']
   del ds[v].attrs['number']
   del ds[v].attrs['molwt']
   del ds[v].attrs['C']

   # Make the units attribute consistent with the units
   # attribute from the GEOS-Chem History diagnostics
   # NOTE: There probably is a more Pythonic way to code
   # this, but this will work for sure.
   if 'ug/m3' in ds[v].units:
      ds[v].attrs['units'] = 'ug m-3'
   if 'ug Celsius/m3' in ds[v].units:
      ds[v].attrs['units'] = 'ug C m-3'
   if 'count/cm3' in ds[v].units:
      ds[v].attrs['units'] = 'molec m-3'
```

(continues on next page)

(continued from previous page)

```
    if 'cm/s' in ds[v].units:
        ds[v].attrs['units'] = 'cm s-1'
    if 'count/cm2/s' in ds[v].units:
        ds[v].attrs['units'] = 'molec cm-2 s-1'
    if 'kg/m2s' in ds[v].units:
        ds[v].attrs['units'] = 'kg m-2 s-1'
    if 'kg/m2/s' in ds[v].units:
        ds[v].attrs['units'] = 'kg m-2 s-1'
    if 'kg/s' in ds[v].units:
        ds[v].attrs['units'] = 'kg s-1'
    if 'W/m2' in ds[v].units:
        ds[v].attrs['units'] = 'W m-2'
    if 'm/s' in ds[v].units:
        ds[v].attrs['units'] = 'm s-1'
    if 'Pa/s' in ds[v].units:
        ds[v].attrs['units'] = 'Pa s-1'
    if 'g/kg' in ds[v].units:
        ds[v].attrs['units'] = 'g kg-1'
    if v.strip() == 'TotalOC':
        ds[v].attrs['units'] = 'ug m-3'
    if v.strip() in [ 'HO2concAfterChem']:
        ds[v].attrs['units'] = 'ppb'
    if v.strip() in ['O1DconcAfterChem',
                     'O3PconcAfterChem',
                     'OHconcAfterChem']:
        ds[v].attrs['units'] = 'molec cm-3'
    if v.strip() in ['Loss_CO', 'Prod_CO',
                     'Loss_Ox', 'Prod_Ox', 'Prod_SO4']:
        ds[v].attrs['units'] = 'molec/cm3/s'
    if v.strip() in 'Met_CLDTOPS':
        ds[v].attrs['units'] = 'level'
    if v.strip() in 'Met_PHIS':
        ds[v].attrs['units'] = 'm2 s-1'
    if v.strip() in ['Met_PRECCON', 'Met_PRECTOT']:
        ds[v].attrs['units'] = 'kg m-2 s-1'
    if v.strip() in 'Met_AVGW':
        ds[v].attrs['units'] = 'vol vol-1'
    if v.strip() in 'Met_AIRNUMDEN':
        ds[v].attrs['units'] = 'molec cm-3'
    if v.strip() in ['ProdCOfromCH4', 'ProdCOfromNMVOC']:
        ds[v].attrs['units'] = 'molec cm-3 s-1'

    # Convert these prodloss diagnostics from kg (bpch) to kg/s
    # to be consistent with the GEOS-Chem History diagnostics
    # NOTE: Assume a 1-month interval (
    if v.strip() in ['ProdSO4fromH2O2inCloud',    'ProdSO4fromO3inCloud',
                     'ProdSO4fromO2inCloudMetal', 'ProdSO4fromO3inSeaSalt',
                     'ProdSO4fromHOBrInCloud',    'ProdSO4fromSRO3',
                     'ProdSO4fromSRHObr',         'ProdSO4fromO3s']:
        ds[v].attrs['units'] = 'kg S s-1'
        ds[v] = ds[v] / interval
    if v.strip() in ['LossHNO3onSeaSalt']:
        ds[v].attrs['units'] = 'kg s-1'
        ds[v] = ds[v] / interval

# ----------------------------------------------------------------------
# Edit attributes for coordinate dimensions
```

(continues on next page)

(continued from previous page)

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

# Time
ds['time'].attrs['long_name'] = 'time'
ds['time'].attrs['units'] = \
   'hours since {} 00:00:00.00 UTC'.format(datestr)
ds['time'].attrs['calendar'] = 'standard'
ds['time'].attrs['axis'] = 'T'

# "lon", "lat", "lev"
ds['lon'].attrs['axis'] = 'X'
ds['lat'].attrs['axis'] = 'Y'
ds['lev'].attrs['axis'] = 'Z'
ds['lev'].attrs['units'] = 'level'

# Global title
ds.attrs['title'] = 'Created by bpch2nc.py'
ds.attrs['conventions'] = 'COARDS'
ds.attrs['references'] = 'www.geos-chem.org; wiki.geos-chem.org'

# ----------------------------------------------------------------
# Create the netCDF file
# ----------------------------------------------------------------
ds.to_netcdf(ncfile)
```

CHAPTER

TWELVE

# REPORT A PROBLEM OR REQUEST A FEATURE

If you encounter an error when using GCPy or if any documentation is unclear, you should open a new issue on the GCPy Github page. Pre-defined templates exist for asking a question or reporting a bug / issue.

We are open to adding new functionality to GCPy as requested by its userbase. Some requested functionality may be better suited to example scripts rather than direct code additions to GCPy. In that case, we can add examples to the *Example Scripts* section of this ReadTheDocs site.

CHAPTER

# THIRTEEN

# CONTRIBUTE TO GCPY

We welcome new code additions to GCPy in the form of pull requests. If you have an example you would like to add to this ReadTheDocs site, you can add it to the `examples` folder in the GCPy repository and submit a pull request with this added file. If you would like to suggest changes to the documentation on this site, you can do so by describing your changes in a Github issue or by directly editing the source ReST files included in the GCPy repository and submitting a pull request with your changes.

We do not currently have an automated testing pipeline operational for GCPy. We ask that you test any changes by plotting / tabling relevant diagnostics using the `run_benchmark.py` plotting scripts included in the `benchmark` folder of the repository, then verifying your results against the results of the same script using an unchanged version of GCPy. Any further testing before finalizing your pull request is greatly appreciated.

CHAPTER

FOURTEEN

# EDITING THESE DOCS

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

## 14.1 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`.

## 14.2 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.

## 14.3 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

FIFTEEN

# 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`
   - `benchmark/run_benchmark.py`
   - `benchmark/modules/run_1yr_fullchem_benchmark.py`
   - `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

## Symbols



## A



## B



## C

## D

## E



## F



## G



## I



## L



## M



## N



## O



## P

## R



## S



## T



## U



## V

## W



## X



## Y