cubedsphere

Release 0.1

Aaron Schneider

May 10, 2022

CONTENTS

Library for post processing of MITgcm cubed sphere data

CHAPTER

ONE

CAPABILITIES

- regrid cubed sphere datasets using xESMF and xgcm
- open datasets created by the mnc package (depreciated)
- open datasets using xmitgcm (needs latest version)
- plot original cubed sphere data
- some more small utilities
- more to come…

1.1 Installation

Warning: You need OSX or Linux to use this package. ESMPy (the dependency that handles the regridding) does not work with Windows.

Note: Currently `xgcm==0.5.2` is required. You will get errors if you use a different version``

1.1.1 Preparation

Create conda environment:

```
conda create -n mitgcm
```

Activate environment:

```
conda activate mitgcm
```

1.1.2 prepackaged installation

Install cubedsphere:

```
conda install -c conda-forge cubedsphere
```

Update `xmitgcm` (needs latest version from github repo):

```
pip install git+https://github.com/MITgcm/xmitgcm.git
```

1.1.3 Alternative: Installation of development version

Clone the repository:

```
git clone https://github.com/AaronDavidSchneider/cubedsphere.git
cd cubedsphere
```

Install dependencies:

```
conda install -c conda-forge xesmf esmpy xgcm xmitgcm matplotlib-base xarray
```

Update `xmitgcm` (needs latest version from github repo):

```
pip install git+https://github.com/MITgcm/xmitgcm.git
```

Install cubedsphere:

```
pip install -e .
```

You can now import the cubedsphere package from everywhere on your system

1.2 Usage

1.2.1 Reading Data

We can either read mds data using

`cubedsphere.`**`open_ascii_dataset`**`(`*outdir*, *return_grid=True*, ***kwargs*`)`

Wrapper that opens simulation outputs from standard mitgcm outputs.

Parameters

outdir: string Output directory

return_grid: Boolean Return a grid generated with xmitgcm.get_grid_from_input

****kwargs** everything else that is passed to xmitgcm

Returns

ds: xarray Dataset Dataset of simulation output

grid: xarray Dataset Only if return_grid is True Grid generated with xmitgcm.get_grid_from_input.

Warning: You need to use `useSingleCpuIO=.TRUE.` if you want to use ascii files (the default MITgcm output)

We can also read NETCDF data which has been outputed from the mnc package (depreciated).

`cubedsphere.open_mnc_dataset(`*outdir*, *iternumber*, *fname_list=['state']*`)`

Wrapper that opens simulation outputs from mnc outputs. NOT TESTED.

Parameters

outdir: string Output directory

iternumber: integer iteration number of output file

fname_list: list List of NetCDF file prefixes to read (no need to specify grid files here)

Returns

ds: xarray Dataset Dataset of simulation output

1.2.2 Regridding

`class cubedsphere.Regridder(`*ds*, *cs_grid*, *input_type='cs'*, *d_lon=5*, *d_lat=4*, *concat_mode=False*, *filename='weights'*, *method='conservative'*, ***kwargs*`)`

Class that wraps the xESMF regridder for cs geometry.

Only two methods are possible with the cs geometry: conservative when using concat_mode=False (requires lon_b to have different shape from lon and lat_b from lat) or nearest_s2d when using concat_mode=True. Conservative regridding should be used if possible!

Notes

You can find more examples in the examples directory

Examples

```
>>> import cubedsphere as cs  # import cubedsphere
>>> outdir = "../run"  # specify output directory
>>> # open Dataset
>>> ds_ascii, grid = cs.open_ascii_dataset(outdir_ascii, iters='all', prefix = ["T",
↪"U","V","W"])
>>> # regrid dataset
>>> regrid = cs.Regridder(ds_ascii, grid)
>>> ds_reg = regrid()
```

Attributes

regridder: list or xESMF regrid object (contains) the initialized xESMF regridder

grid: dict output grid

`__init__(`*ds*, *cs_grid*, *input_type='cs'*, *d_lon=5*, *d_lat=4*, *concat_mode=False*, *filename='weights'*, *method='conservative'*, ***kwargs*`)`

Build the regridder. This step will create the output grid and weight files which will then be used to regrid the dataset.

Parameters

ds: xarray DataSet Dataset to be regridded. Dataset must contain input grid information.

cs_grid: xarray DataSet Dataset containing cubedsphere grid information. If input_type==”cs”: Input grid. Required! If input_type==”ll”: Output cs grid. Required!

input_type: string Needs to be “cs” or “ll”. Will result in an error if something else is used here. If “cs”: input=cs grid -> regrid to longitude lattitude (ll) If “ll”: input=ll grid -> regrid to cs grid

d_lon: integer Longitude step size, i.e. grid resolution. Only used if input_type==”cs”.

d_lat: integer Latitude step size, i.e. grid resolution. Only used if input_type==”cs”.

concat_mode: boolean use one regridding instance instead of one regridder for each face. Only used if input_type==”cs”.

filename: string filename for weights (weights will be name filename(+ _tile{i}).nc)

method: string Regridding method. See xe.Regridder for options.

kwargs Optional parameters that are passed to xe.Regridder (see xe.Regridder for options).

`__call__`(***kwargs*)

Wrapper that carries out the regridding from cubedsphere to latlon.

Parameters

Returns

ds: xarray DataSet regridded Dataset

1.2.3 Create XGCM grids

`cubedsphere.init_grid_CS`(*ds=None*, *grid_dir=None*, ***kwargs*)

Init a xgcm grid for a cubedsphere dataset. Useful for raw datasets.

Parameters

grid_dir: string direction where the grid can be found (optional)

ds: xarray DataSet dataset that contains the grid (optional)

Returns

grid: xgcm.grid xgcm grid

`cubedsphere.init_grid_LL`(*ds=None*, *grid_dir=None*, ***kwargs*)

Init a xgcm grid for a latlon dataset. Useful for regridded datasets

Parameters

grid_dir: string direction where the grid can be found (optional)

ds: xarray DataSet dataset that contains the grid (optional)

Returns

grid: xgcm.grid xgcm grid

1.2.4 Plotting

cubedsphere.**plotCS**(*dr*, *ds*, *mask_size=None*, ***kwargs*)

A quick way to plot cubed-sphere data of resolution 6*N*N. Wrapping plotCS_quick_raw to work with xarray objects

Parameters

dr: xarray DataArray The dimensions must be (y,x)

ds: xarray DataSet (the parent DataSet of dr) Must contain XC, YC as coordinate variables.

mask_size: None or int The overlap size of individual tiles. If None is chosen one might likely experience issues

****kwargs** Other keyword arguments such as cmap will be passed to plotCS_quick_raw() and subsequently to plt.pcolormesh()

Returns

ph: list List of mappabales

cubedsphere.**overplot_wind**(*ds_reg*, *U*, *V*, *stepsize=1*, ***kwargs*)

Quick and dirty function for overplotting wind of a regridded dataset

Parameters

ds_reg: xarray DataSet regridded dataset

stepsize: integer specify the stepsize for which wind arrows should be plotted

1.3 Examples

import cubedsphere and other helper packages

[1]:
```
import numpy as np
import cubedsphere as cs

import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import cartopy.crs as ccrs  # optional, only needed for nicer projections
```

Specify folder of simulationdata

[2]:
```
outdir_ascii = "/Volumes/EXTERN/Simulations/exorad/new_run/paper_runs/WASP-43b/run/"
```

1.3.1 Standard conservative regridding

Load data and regrid data

[3]:
```
# open Dataset using xmitgcm (see docs for xmitgcm.open_mdsdataset for more details)
ds_ascii, grid = cs.open_ascii_dataset(outdir_ascii, iters=[41472000], prefix = ["T","U",
↪"V","W"])
# regrid dataset
regrid = cs.Regridder(ds_ascii, grid)
ds = regrid()
```

(continues on next page)

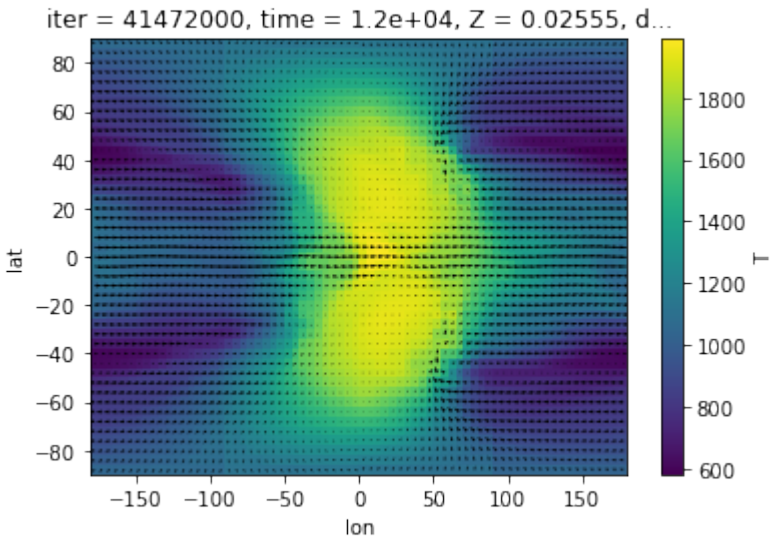
(continued from previous page)

```
# (optional) converts wind, temperature and stuff
ds = cs.exorad_postprocessing(ds, outdir=outdir_ascii)
```

```
time needed to build regridder: 0.9600319862365723
Regridder will use conservative method
```

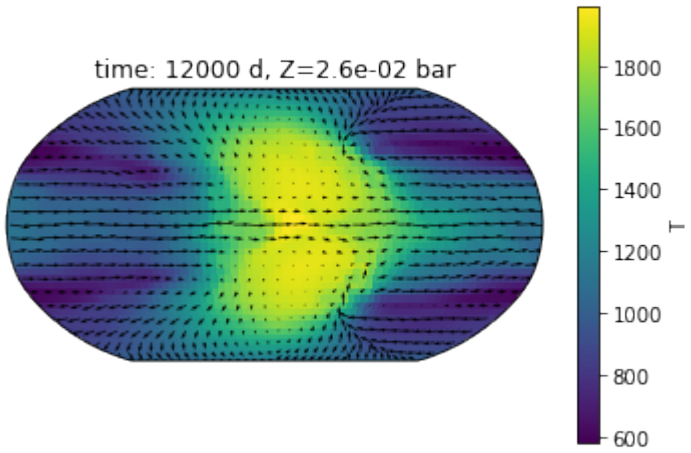
Minimal working example of a plot:

[4]:
```
plt.figure()
# Select horizontal slice at latest time:
data = ds.isel(time=-1,Z=-20)
# Plot temperature:
data.T.plot()
# Overplot winds:
cs.overplot_wind(ds, data.U.values, data.V.values)
plt.show()
```



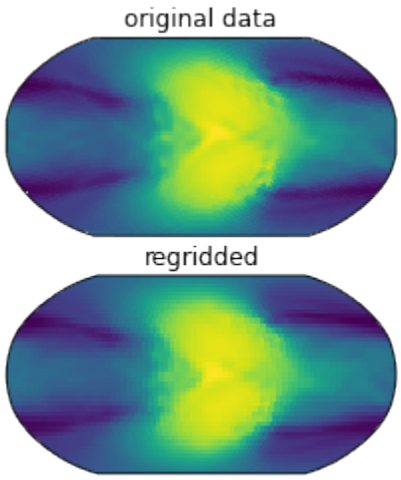
A littlebit nicer with cartopy:

[5]:
```
plt.figure()
ax = plt.axes(projection=ccrs.Robinson())
# Plot temperature:
data.T.plot(transform = ccrs.PlateCarree(), ax=ax)
# Overplot winds:
cs.overplot_wind(ds, data.U.values, data.V.values, ax=ax, transform=ccrs.PlateCarree(),␣
↪stepsize=2)
ax.set_title('time: {:.0f} d, Z={:.1e} bar'.format(data.time.values,data.Z.values))
plt.show()
```



Utilizing the cubedsphere plot to do a comparison with the original (not regridded) data:

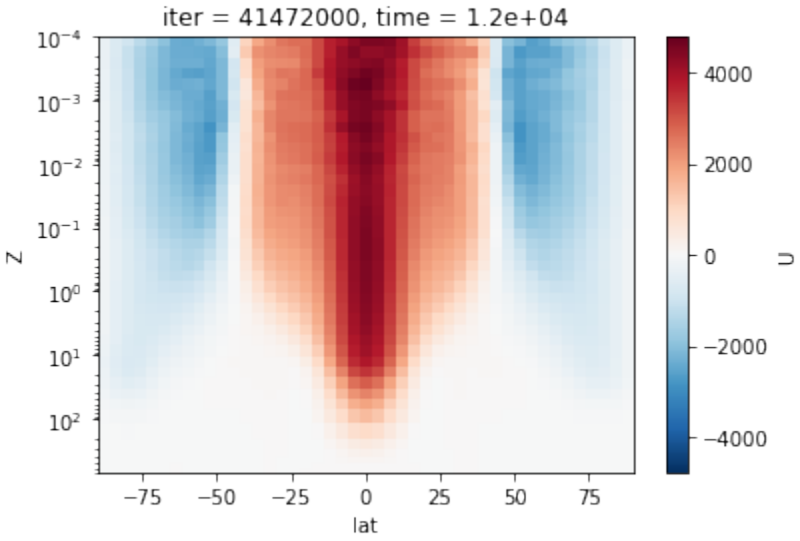
[6]:
```
data_orig = ds_ascii.isel(time=-1,Z=-20)

fig, ax = plt.subplots(2,1, subplot_kw={"projection":ccrs.Robinson()})

# Do the plots
cs.plotCS(data_orig.T, data_orig, transform=ccrs.PlateCarree(), ax = ax[0])
ax[1].pcolormesh(data.lon, data.lat, data.T, transform = ccrs.PlateCarree())

ax[0].set_title('original data')
ax[1].set_title('regridded')

plt.show()
```



1.3.2 Zonal mean plots

For convenience, we will show an easy way on how to create a zonal mean wind plot

[7]:
```
plt.figure()
zmean = ds.U.isel(time=-1).mean(dim='lon')
zmean.plot()
plt.yscale('log')
plt.ylim([700,1e-4])
plt.show()
```



1.3.3 From lon,lat to cubedsphere

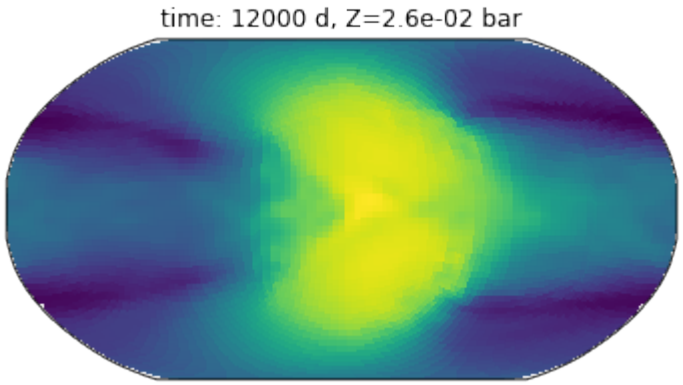
We are now going to perform a second regridding (from already regridded original to cubedsphere)

[8]:
```
# Add some info to the regridded dataset
reg_grid = regrid._build_output_grid(5, 4)
ds["lon_b"] = reg_grid["lon_b"]
ds["lat_b"] = reg_grid["lat_b"]

# Perform back regridding
regrid = cs.Regridder(ds, grid, input_type='ll')
ds_regback = regrid()
```

```
time needed to build regridder: 0.9572968482971191
Regridder will use conservative method
```

Congratulations! `ds_regback` is now again in cubedsphere coordinates!

[9]:
```
plt.figure()
data_reg_back = ds_regback.isel(time=-1, Z=-20)
ax = plt.axes(projection=ccrs.Robinson())
ax.set_title('time: {:.0f} d, Z={:.1e} bar'.format(data.time.values,data.Z.values))
cs.plotCS(data_reg_back.T, data_reg_back, transform=ccrs.PlateCarree(), ax = ax)
plt.show()
```



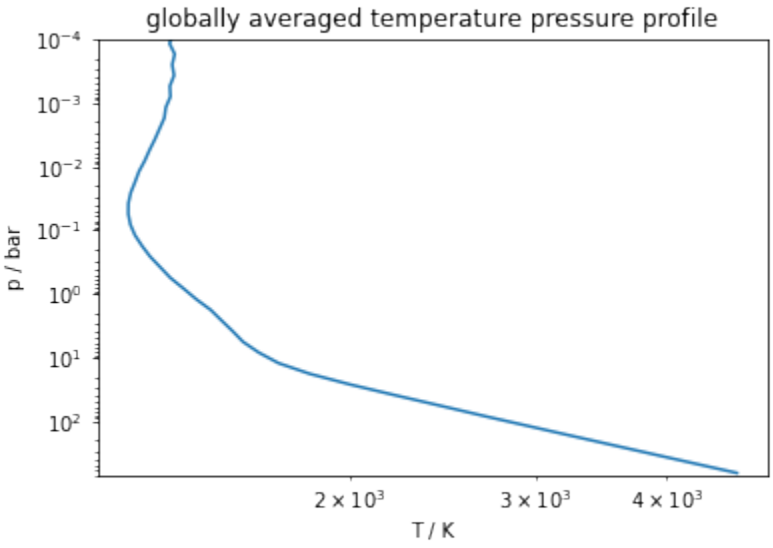
1.3.4 Calculate global averages

a globally averaged quatity $\bar{T}$ can be calculated by

$\bar{T} = \frac{\sum T \cdot \Delta A}{\sum \Delta A}$,

where ΔA is the area of the grid cell.

[10]:
```
plt.figure()
T_global = (ds.T.isel(time=-1)*ds.area_c).sum(dim=['lon','lat'])/ds.area_c.sum(dim=['lon
↪','lat'])
plt.loglog(T_global, ds.Z)
plt.ylim(700,1e-4)
plt.title('globally averaged temperature pressure profile')
plt.ylabel('p / bar')
plt.xlabel('T / K')
plt.show()
```



1.3.5 Open userspecific files

Some packages (e.g., SPARC/MITgcm and exPERT/MITgcm) are not opensource and we therefore need to specify how we have to read the extra output generated by those codes.

This can be easily done using the `extra_variables` keyword, passed to `open_mdsdataset` from `cs.open_ascii_dataset`.

Extra variables can also be added to the codebase of the `cubedsphere` package. You can find already added `exPERT/MITgcm` variables here.

Note: Click here to see the options for all `kwargs` used to open datasets with `xmitgcm`.

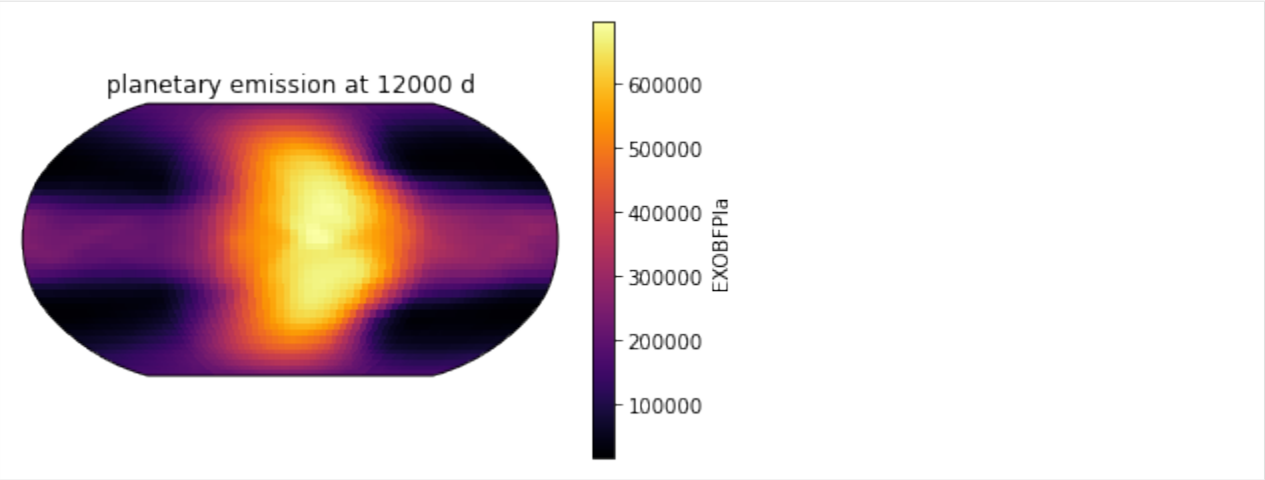
We will now show an example of how we can load and plot the bolometric emission fluxes generated from the exPERT/MITgcm output.

[11]:
```
# Note: Not needed, since already part of cubedsphere package, this is shown only to␣
↪demonstrate how it works
extra_variables = dict(EXOBFPla=dict(dims=['k_p1', 'j', 'i'],
                                     attrs=dict(standard_name='EXOBFPla', long_
↪name='Bolometric Planetary Flux',
                                                units='W/m2')))
```

[12]:
```
# open Dataset using xmitgcm (see docs for xmitgcm.open_mdsdataset for more details)
ds_ascii, grid = cs.open_ascii_dataset(outdir_ascii, iters=[41472000], prefix = [
↪"EXOBFPla"], extra_variables=extra_variables)
# regrid dataset
regrid = cs.Regridder(ds_ascii, grid)
ds = regrid()
# (optional) converts wind, temperature and stuff
ds = cs.exorad_postprocessing(ds, outdir=outdir_ascii)
```

```
could not rename, got error: cannot rename 'T' because it is not a variable or dimension␣
↪in this dataset
time needed to build regridder: 0.9598851203918457
Regridder will use conservative method
```

[13]:
```
# Plot planetary emission:
plt.figure()
ax = plt.axes(projection=ccrs.Robinson())
ds.EXOBFPla.isel(time=-1, Zp1=-1).plot(transform = ccrs.PlateCarree(), ax=ax, cmap=plt.
↪get_cmap('inferno'))
ax.set_title('planetary emission at {:.0f} d'.format(data.time.values))
plt.show()
```



1.4 Exorad

1.4.1 Convert data

We can convert exorad outputs using

cubedsphere.exorad.**exorad_postprocessing**(*ds*, *outdir=None*, *datafile=None*, *convert_to_bar=True*, *convert_to_days=True*)

Preliminaray postprocessing on exorad dataset. This function converts the vertical windspeed from Pa into meters and saves attributes to the dataset.

Parameters

ds: Dataset dataset to be extended

outdir: string directory in which to find the data file (following the convention f'{outdir}/data')

datafile: string alternatively specify datafile directly

convert_to_bar: (Optional) bool convert vertical pressure dimension to bar

convert_to_days: (Optional) bool convert time dimension to days

Returns

ds: Dataset to be returned

CHAPTER

TWO

CREDITS

Many of the methods come from: https://github.com/JiaweiZhuang/cubedsphere

I would especially like to thank @rabernat for providing xgcm and @JiaweiZhuang for providing xESMF.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

INDEX

Symbols

E

I

O

P

R