matteodefelice.name

How to create a multi-model of C3S Seasonal Forecasts with xarray and cfgrib | Matteo De Felice

Matteo De Felice

6-7 minutes

NOTE: this post has been updated. The previous code was based on the conversion of the GRIB files into NetCDF, which introduces unfortunately some issues.

Among the data products of the Copernicus Climate Change (C3S) available through the Climate Data Store, there is a collection of seasonal forecasts, from the 13th November consisting of five different models (ECMWF, UK Met Office, Meteo-France, DWD and CMCC).

One of the interesting things you can do with multiple climate models is to combine them into a multi-model ensemble. With Python and xarray this is very simple. However, although the CDS has made the retrieval of climate data more user-friendly than in the past (nice web interface, API in Python, documentation, etc.), there are still known issues and some undocumented things that must be taken into account.

In this post I will show how to combine a set of seasonal forecasts, retrieved from the CDS, in the simplest way. For this example I have used four out of the five forecasting systems currently available on C3S, excluding the CMCC because it is still not available for all the starting dates.

Among the models there could be subtle differences on the regular grids used, then I have regridded the data using the `xesmf` package.

Then, those are the modules I have used for this piece of code (based on Python 3.6)

```
import cfgrib
import xarray as xr
import numpy as np
import xesmf as xe
```

The CDS provides the data in GRIB format a format that is not natively supported by `xarray`. However, the module `cfgrib` –

through the ECMWF `ecCodes` library — provides high-level API and enables an additional backend for xarray to read GRIB files. that in principle are easily accessible with `cfgrib`. Another solution would be to convert the data in NetCDF with the command `grib_to_netcdf` from the terminal (for examples installing the `eccodes` package with Anaconda), but however – as stated at the beginning of this post – can lead to issues and erroneous conversions.

To regrid the data I have defined a common grid (here a ²⁄₂ regular grid):

```
ds_out = xr.Dataset({'lat': (['lat'],
np.arange(-89, 89, 2)),
                     'lon': (['lon'], np.arange(0,
360, 2))})
```

Now I can load all the four GRIB files (each one providing the monthly statistics of mean sea-level pressure with the December starting date). To save a bit of time I have selected only the first lead-time (with `isel`).

```
d = xr.open_dataset('/my/path/to/ECMWF-mslp-S12-
L1-4.grib', engine = 'cfgrib')
d = d.rename({'latitude':'lat', 'longitude':
'lon'}).isel(step = 0)
regridder = xe.Regridder(d, ds_out, 'bilinear')
d_ecmwf = regridder(d.msl)
```

Similarly for the other three models:

```
d = xr.open_dataset('/opt/data/C3S/UKMO-mslp-S12-
L1-4.grib', engine = 'cfgrib', backend_kwargs=
{'filter_by_keys':
{'longitudeOfFirstGridPointInDegrees': 0.0}})
d = d.rename({'latitude':'lat', 'longitude':
'lon'}).isel(step = 0)
regridder = xe.Regridder(d, ds_out, 'bilinear')
d_ukmo = regridder(d.msl)
```

```
d = xr.open_dataset('/opt/data/C3S/DWD-mslp-S12-
L1-4.grib', engine = 'cfgrib')
d = d.rename({'latitude':'lat', 'longitude':
'lon'}).isel(step = 0)
regridder = xe.Regridder(d, ds_out, 'bilinear')
d_dwd = regridder(d.msl)
```

```
d = xr.open_dataset('/opt/data/C3S/MF-mslp-S12-
L1-4.grib', engine = 'cfgrib', backend_kwargs=
{'filter_by_keys':{'numberOfPoints': 64800}})
d = d.rename({'latitude':'lat', 'longitude':
'lon'}).isel(step = 0)
regridder = xe.Regridder(d, ds_out, 'bilinear')
d_mf = regridder(d.msl)
```

To understand the meaning of the `backend_kwargs` parameter, I would strongly suggest reading the Advanced Usage section of the cfgrib documentation and the Issues on the cfgrib repository.

The rename is needed by the regridding functions. In this example I have used seasonal monthly data but in principle the same approach can be used for the hourly version, you would just need more memory (the files used in this example are between 300 and 600 MB) and probably you would consider using dask with xarray.

Now I can merge the models using the concatenation function from xarray. For this operation you have to specify along which dimension you want to concatenate along the data cubes and, very important, the dimensions must be consistent, otherwise everything will become very messy (try for example to concatenate the models *without* the regridding).

```
c3s_multimodel = xr.concat([d_ecmwf, d_ukmo,
d_dwd, d_mf], dim = 'number')
nmembers = len(c3s_multimodel.number.values)
c3s_multimodel['number'] = np.arange(0, nmembers)
c3s_multimodel
```

```
<xarray.DataArray 'msl' (number: 108, time: 23,
lat: 89, lon: 180)>
array([[[[100132.3125, ..., 100129.5625],
         ...,
         [101353.0625, ..., 101352.6875]],

        ...,

        [[ 99986.1875, ...,  99981.9375],
         ...,
         [101415.5625, ..., 101413.3125]]],


       ...,


       [[[100475.    , ..., 100472.375 ],
         ...,
         [102115.875 , ..., 102118.    ]],
        ...,

        [[101482.5   , ..., 101479.5   ],
         ...,
         [100651.625 , ..., 100655.125 ]]]])
Coordinates:
  * time        (time) datetime64[ns] 1993-12-01
1994-12-01 ... 2015-12-01
  * lon         (lon) int64 0 2 4 6 8 10 12 14 ...
346 348 350 352 354 356 358
  * lat         (lat) int64 -89 -87 -85 -83 -81
-79 -77 ... 75 77 79 81 83 85 87
    step        timedelta64[ns] 31 days
    surface     int64 0
  * number      (number) int64 0 1 2 3 4 5 6 7 ...
101 102 103 104 105 106 107
    valid_time  (number, time) datetime64[ns]
1994-01-01 ... 2016-01-01
Attributes:
    regrid_method:  bilinear
```

Here we are: we have a multi-model ensemble of four seasonal forecasts model with 108 members.

As you can see, the entire workflow is very linear and easy to code. Before xarray and python you could do the same using a set of powerful tools like `nco`, `cdo` and the various GRIB utilities, but the possibility to have the entire workflow **starting from the download** (through the CDS API) reproducible and into the same flexible & multi-platform environment is definitely priceless.