Plotting with Python

This tutorial is the third part of a series that shows how to process climate model data on distributed High Performance Computing (HPC) environments with state of the art python libraries. Visit the **Part 2** and **Part 1** to see how to read and remap the data.

In this tutorial we are going to plot the data that has been processed previously. Like in any other tutorial lets import the libraries we are going to use first using the *python 3 unstable* kernel:

```
from getpass import getuser # Libaray to copy things
from pathlib import Path # Object oriented libary to deal with paths

from cartopy import crs as ccrs # Cartography library
from matplotlib import pyplot as plt # Standard Plotting library
import numpy as np # Standard array library
import pandas as pd # Libary to work with labeled data frames and time
series
import seaborn as sns # Makes plots more beautiful
import xarray as xr # Libary to work with labeled n-dimensional data and
dask
```

In **Part 2** of this tutorial we remapped the output of global storm resolving model simulations to a regular latitude-longitude grid and save the averages over time and space to file. We are now going to read the saved files and display the content:

```
scratch_dir = Path('/scratch') / getuser()[0] / getuser() # Define the users
scratch dir
out_file = Path(scratch_dir) / 'dpp0016_PreProc.nc'

time_mean = xr.open_dataset(out_file, group='time_mean').load()
field_mean = xr.open_dataset(out_file, group='field_mean').load()
```

```
time_mean
```

```
<xarray.Dataset>
Dimensions:  (lat: 1800, lon: 3600)
Coordinates:
  * lon      (lon) float64 -179.9 -179.8 -179.8 -179.6 ... 179.8 179.9 180.0
  * lat      (lat) float64 -89.95 -89.85 -89.75 -89.65 ... 89.75 89.85 89.95
Data variables:
    ts       (lat, lon) float32 221.04167 221.04166 ... 257.14673 257.14676
    rsus     (lat, lon) float32 55.74567 55.745674 ... 56.61574 56.615986
    rlus     (lat, lon) float32 136.61485 136.6148 ... 253.50243 253.50249
    rlds     (lat, lon) float32 108.62206 108.622185 ... 232.81358 232.81357
    rsds     (lat, lon) float32 69.93069 69.930695 69.9307 ... 84.0771
84.07749
    hfls     (lat, lon) float32 0.0656544 0.06563331 ... -6.8061843
-6.806321
    hfss     (lat, lon) float32 10.846811 10.845357 ... 0.60061556
0.60098815
```

```
Attributes:
    CDI:                  Climate Data Interface version 1.8.3rc
(http://mpim...
    Conventions:          CF-1.6
    number_of_grid_used:  15
    grid_file_uri:
http://icon-downloads.mpimet.mpg.de/grids/public/mp...
    uuidOfHGrid:          0f1e7d66-637e-11e8-913b-51232bb4d8f9
    title:                ICON simulation
    institution:          Max Planck Institute for Meteorology/Deutscher
Wett...
    source:               git@gitlab.dkrz.de:icon/icon-
aes.git@b582fb87edbd30...
    history:              /work/mh0287/k203123/GIT/icon-aes-
dyw_albW/bin/icon...
    references:           see MPIM/DWD publications
    comment:              Sapphire Dyamond (k203123) on m21322 (Linux
2.6.32-...
```

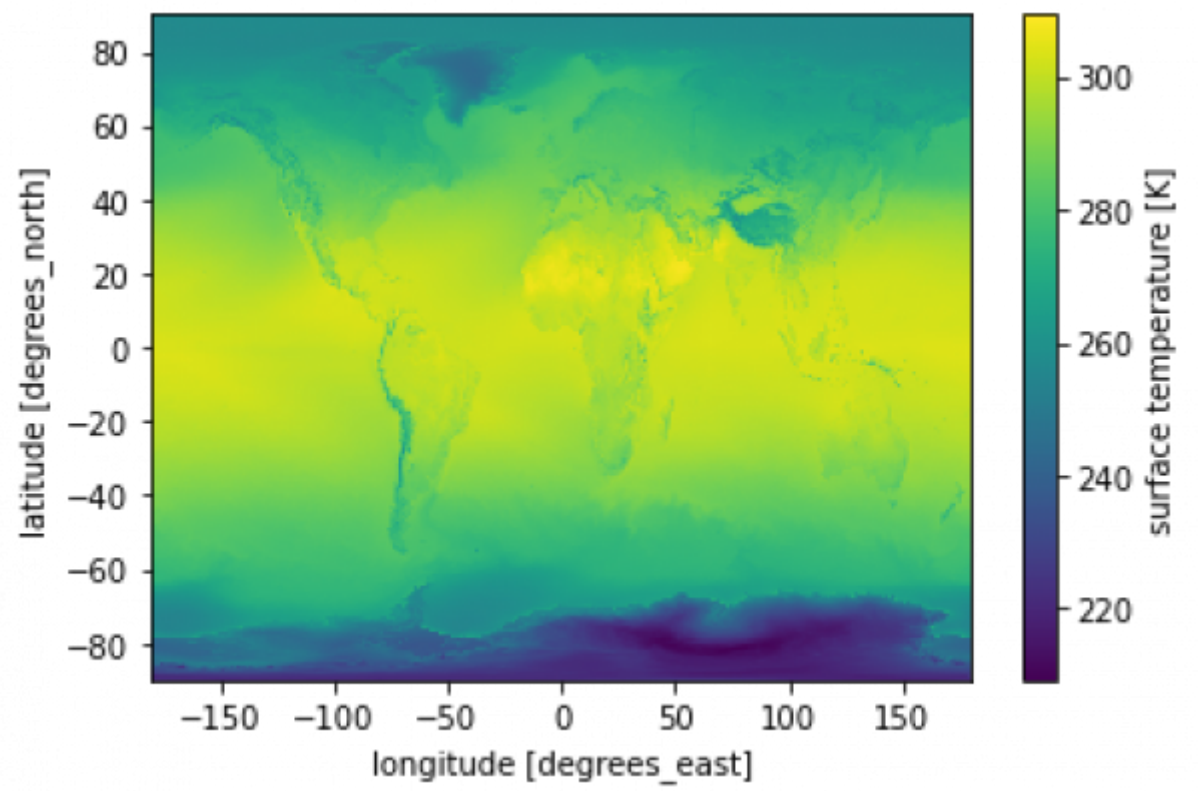
```
field_mean
```

```
<xarray.Dataset>
Dimensions:  (time: 304)
Coordinates:
  * time     (time) datetime64[ns] 2020-02-01 2020-02-02 ... 2020-11-30
Data variables:
    ts       (time) float32 286.06442 285.98172 ... 287.01483 287.09506
    rsus     (time) float32 35.026955 35.132553 ... 38.266792 37.940987
    rlus     (time) float32 387.74957 387.44562 ... 392.57364 392.88156
    rlds     (time) float32 335.21866 334.84085 334.0962 ... 340.49408
341.90875
    rsds     (time) float32 200.44205 200.92815 ... 202.39125 201.83057
    hfls     (time) float32 -76.36643 -77.96001 ... -89.82848 -84.05612
    hfss     (time) float32 -22.51513 -23.409267 ... -24.214497 -22.375736
Attributes:
    CDI:                  Climate Data Interface version 1.8.3rc
(http://mpim...
    Conventions:          CF-1.6
    number_of_grid_used:  15
    grid_file_uri:
http://icon-downloads.mpimet.mpg.de/grids/public/mp...
    uuidOfHGrid:          0f1e7d66-637e-11e8-913b-51232bb4d8f9
    title:                ICON simulation
    institution:          Max Planck Institute for Meteorology/Deutscher
Wett...
    source:               git@gitlab.dkrz.de:icon/icon-
aes.git@b582fb87edbd30...
    history:              /work/mh0287/k203123/GIT/icon-aes-
dyw_albW/bin/icon...
```

```
    references:           see MPIM/DWD publications
    comment:              Sapphire Dyamond (k203123) on m21322 (Linux
2.6.32-...
```

xarray has a powerful plot functionality that can be used to quickly plot and inspect the data:

```
time_mean['ts'].plot()
```

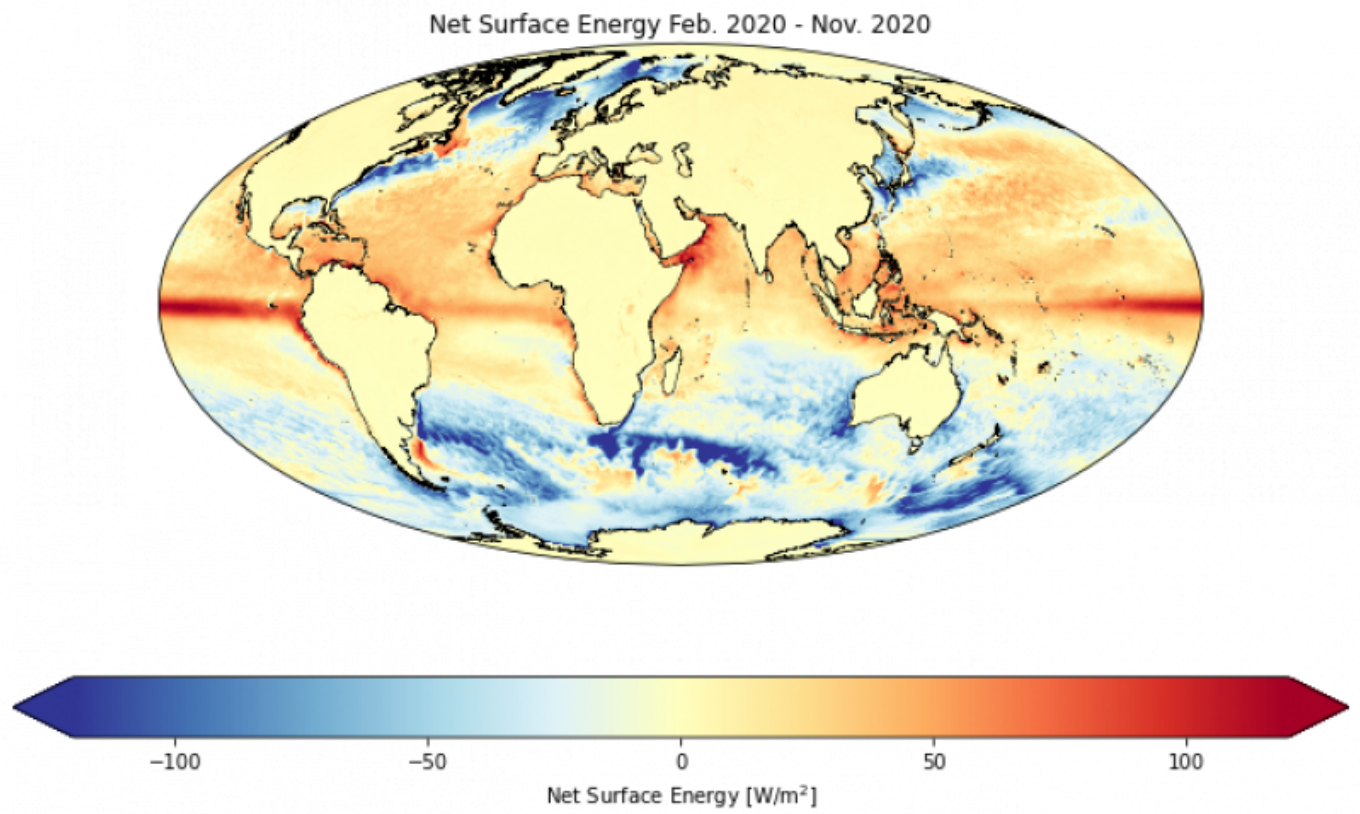


These plots are not very nice but we can make them much nicer by using cartopy to draw maps. Xarray supports subplots and fiddeling with the colorbar. Let's create a nicer plot of net surface energy using the cartopy library. Net surface energy is defined as the sum of net surface radiation, latent and surface heat fluxes:

```
time_mean['surf_energy'] = (time_mean['rsds'] - time_mean['rsus']) \
                         + (time_mean['rlds'] - time_mean['rlus']) \
                         + (time_mean['hfls'] + time_mean['hfss'])
```

```
proj = ccrs.Mollweide(50.) # Create Mollweide projections
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection=proj)
plot = time_mean['surf_energy'].plot(
                        ax=ax,
                        transform=ccrs.PlateCarree(),
                        cmap='RdYlBu_r',
                        vmin=-120,
                        vmax=120,
                        cbar_kwargs={'label': 'Net Surface Energy [W/m$^2$]',
                                     'extend': 'both',
```

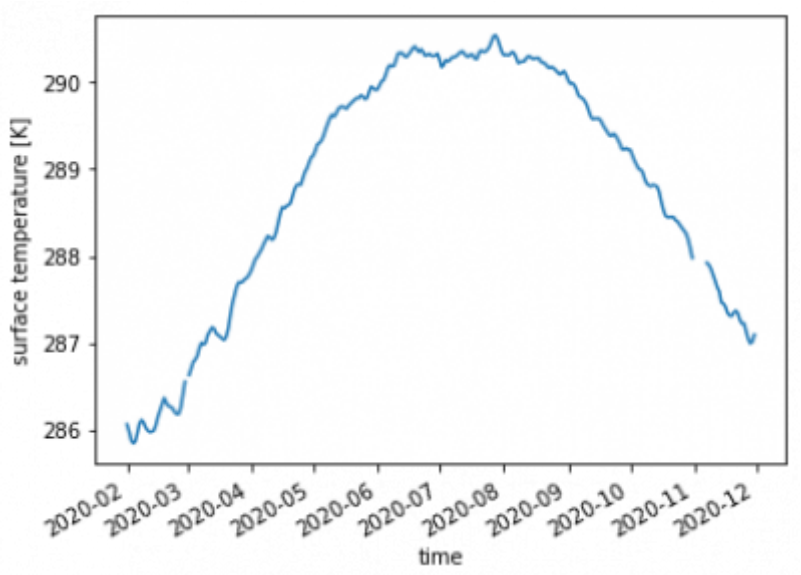
```
                                        'shrink': .8,
                                        'orientation': 'horizontal'},

                          )
ax.coastlines(resolution='10m', lw=0.51)
ax.set_title(f'Net Surface Energy Feb. 2020 - Nov. 2020')
_ = fig.subplots_adjust(left=0.01, right=0.98, hspace=0, wspace=0, top=0.9,
bottom=0.25)
```



Time series plots can be done in a similar fashion:

```
field_mean['ts'].plot()
```

```
[<matplotlib.lines.Line2D at 0x2aec3b6f5d90>]
```



Let's make this plot a little more pretty. The Seaborn library provides a nice interface for that:

```
col_blind = sns.color_palette("colorblind", 10)
sns.set_style('ticks')
sns.set_palette(col_blind)
sns.set_context("notebook", font_scale=1., rc={"lines.linewidth": 3.5})
```

Now let's create two plots showing the time series and zonal averages next to each other:

```
fig, axs = plt.subplots(1, 2,figsize=(12, 6), sharey=False)
_ = (field_mean['ts'] - 273.15).plot.line(x='time', ax=axs[0],
add_legend=False)
axs[0].set_ylabel('Surface Temperature [$^\\circ$C]')
_ = (time_mean['ts']-273.15).mean(dim='lon').plot.line(x='lat', ax=axs[1])
axs[-1].set_ylabel('')
axs[-1].set_xlabel('Latitude [$^\\circ$N]')
fig.suptitle('Surface temperature Feb 2020 - Nov 2020')
fig.subplots_adjust(left=0.1, right=0.99, wspace=0.2, top=.95, bottom=0.3)
sns.despine()
```