

data_processing

March 23, 2022

1 Exercise: CMIP6 - Plot Time Series of annual means

In this task, the global annual means of the CMIP6 variable `tas` are to be calculated and plotted.

Since one time series alone would be quite boring we want to generate a total of 4 time series of the scenarios

SSP 1-2.6

SSP 2-4.5

SSP 3-7.0

SSP 5-8.5

For more information about the scenarios see

<https://www.dkrz.de/en/communication/climate-simulations/cmip6-en/the-ssp-scenarios>

The data are stored in several separate files and are to be merged with CDO and then their field mean average calculated.

1.0.1 In the exercise you will:

1. Learn to use the netCDF input files
2. Process the data with CDO or xarray
3. Visualize the processed data using matplotlib
4. Create a plot for a poster using seaborn

An example plot of the scenarios of some more ensembles and their mean values can be found on the DKRZ page for CMIP6

<https://www.dkrz.de/de/kommunikation/klimasimulationen/cmip6-de/cmip6-aktivitaeten-am-dkrz-ueberblick>

1.0.2 Input Data

Historical: `/pool/data/CMIP6/data/CMIP/MPI-M/MPI-ESM1-2-LR/historical/r1i1p1f1/Amon/tas/gn/v20190710/`

Scenario SSP 1-2.6: `/pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp126/r1i1p1f1/Amon/tas/gn/v20190710/`

Scenario SSP 2-4.5: /pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp245/r1i1p1f1/Amon/tas/gn/v20190710/

Scenario SSP 3-7.0: /pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp370/r1i1p1f1/Amon/tas/gn/v20190710/

Scenario SSP 5-8.5: /pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp585/r1i1p1f1/Amon/tas/gn/v20190710/

1.0.3 CDO hints:

1. Merge files along time (mergetime)
2. Compute the annual mean (yearmean)
3. Compute the field means (fldmean)
4. Take care of the shapes
5. Important: clean temporary files

1.0.4 Xarray hints:

1. Open multiple files at once
2. Compute the weights (http://xarray.pydata.org/en/stable/examples/area_weighted_temperature.html)
3. Compute the annual mean (resample)
4. Compute the field means (mean)

1.1 Solution

1.1.1 Import CDO module

```
[1]: from cdo import *  
  
cdo = Cdo()
```

1.1.2 Import Matplotlib module

```
[2]: from matplotlib import pyplot as plt  
  
%matplotlib inline
```

1.1.3 Let's see what is in the ssp126 data directory

There are two ways to do it:

```
%bash
```

```
ls /pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp126/r1i1p1f1/Amon/tas/gn/v20190710/
```

or

```
!ls /pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp126/r1i1p1f1/Amon/tas/gn/v20190710/
```

The result is the same.

```
[3]: %%bash
ls /pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp126/r1i1p1f1/Amon/
↪tas/gn/v20190710/
```

```
tas_Amon_MPI-ESM1-2-LR_ssp126_r1i1p1f1_gn_201501-203412.nc
tas_Amon_MPI-ESM1-2-LR_ssp126_r1i1p1f1_gn_203501-205412.nc
tas_Amon_MPI-ESM1-2-LR_ssp126_r1i1p1f1_gn_205501-207412.nc
tas_Amon_MPI-ESM1-2-LR_ssp126_r1i1p1f1_gn_207501-209412.nc
tas_Amon_MPI-ESM1-2-LR_ssp126_r1i1p1f1_gn_209501-210012.nc
```

1.2 Input files

For the sake of simplicity, we take the full path of the directories.

```
[4]: data_ssp126 = '/pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp126/
↪r1i1p1f1/Amon/tas/gn/v20190710/tas_*.nc'
data_ssp245 = '/pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp245/
↪r1i1p1f1/Amon/tas/gn/v20190710/tas_*.nc'
data_ssp370 = '/pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp370/
↪r1i1p1f1/Amon/tas/gn/v20190710/tas_*.nc'
data_ssp585 = '/pool/data/CMIP6/data/ScenarioMIP/MPI-M/MPI-ESM1-2-LR/ssp585/
↪r1i1p1f1/Amon/tas/gn/v20190710/tas_*.nc'
```

```
[5]: #cdo.sinfon(input=data_ssp126)
```

1.3 Compute the field mean of the yearly mean

Remember that CDO commands are written from right to left!

We want to do

1. merge all files of a scenario respecting the time dimension
2. compute the yearly mean
3. compute the field mean
4. assign the output time series to a variable of type xarray.DataArray

```
[6]: tas126_ymean = cdo.fldmean(input='-yearmean -mergetime ' + data_ssp126,
↪returnXArray='tas')
tas245_ymean = cdo.fldmean(input='-yearmean -mergetime ' + data_ssp245,
↪returnXArray='tas')
tas370_ymean = cdo.fldmean(input='-yearmean -mergetime ' + data_ssp370,
↪returnXArray='tas')
```

```
tas585_ymean = cdo.fldmean(input='-yearmean -mergetime ' + data_ssp585,
    ↪returnXArray='tas')
```

1.4 Keep an eye on the data shapes

Printing the variables shape shows that we have 3 shapes which is a problem for matplotlib in our case.

```
print(tas126_ymean.shape)
print(tas245_ymean.shape)
print(tas370_ymean.shape)
print(tas585_ymean.shape)
```

```
(86, 1, 1)
(86, 1, 1)
(86, 1, 1)
(86, 1, 1)
```

We can remove the lat and lon dimensions with size 1 using array slicing.

```
data126 = tas126_ymean[:,0,0]
data245 = tas245_ymean[:,0,0]
data370 = tas370_ymean[:,0,0]
data585 = tas585_ymean[:,0,0]
```

To prevent this step use CDO's option `--reduce_dims` within the CDO command line. In addition we use explicitly the data output type float64 for the comparison with the output of the xarray computation.

For example

```
tas126_ymean = cdo.fldmean(options='-b F64 --reduce_dim',
    input='-yearmean -mergetime ' + data_ssp126,
    returnXArray='tas')
```

```
[7]: data126 = cdo.fldmean(options='-b F64 --reduce_dim', input='-yearmean_
    ↪-mergetime ' + data_ssp126, returnXArray='tas')
data245 = cdo.fldmean(options='-b F64 --reduce_dim', input='-yearmean_
    ↪-mergetime ' + data_ssp245, returnXArray='tas')
data370 = cdo.fldmean(options='-b F64 --reduce_dim', input='-yearmean_
    ↪-mergetime ' + data_ssp370, returnXArray='tas')
data585 = cdo.fldmean(options='-b F64 --reduce_dim', input='-yearmean_
    ↪-mergetime ' + data_ssp585, returnXArray='tas')
```

1.5 Define x-axis

The scenario time range is year 2015 to year 2100 and we define a list variable x with this value range.

```
[8]: x = range(2015,2101,1)
```

1.6 Create the time series plot of scenarios

In the next step

1. define the plotting figure and axis
2. draw all 4 scenario time series
3. draw grid lines
4. add a title string
5. define the axis titles
6. add a legend

```
[9]: fig, ax = plt.subplots(figsize=(8,4))

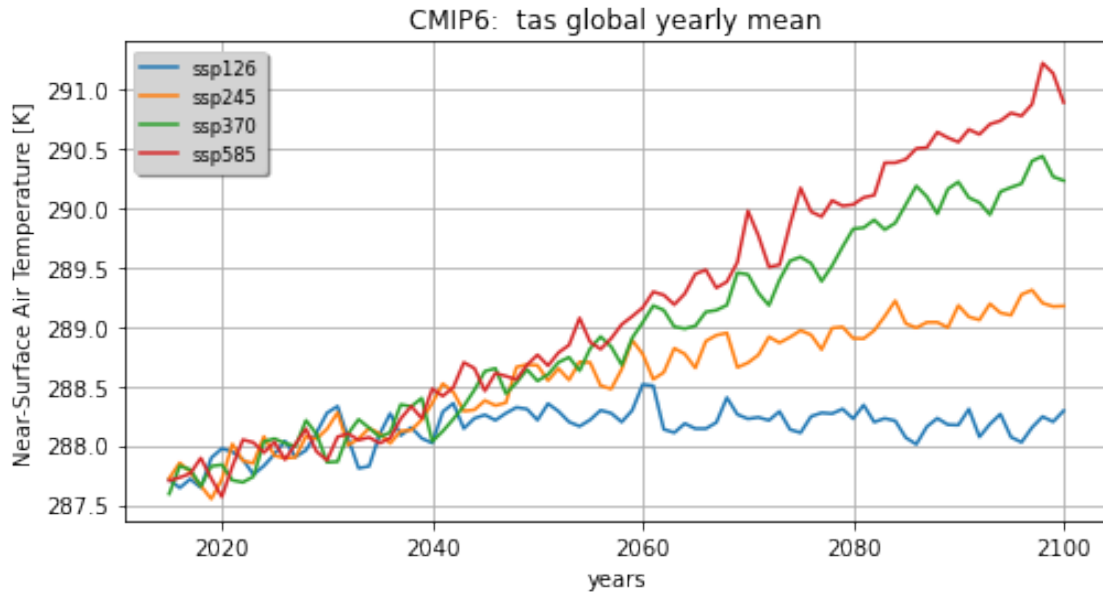
# draw scenario time series
ax.plot(x, data126, label='ssp126')
ax.plot(x, data245, label='ssp245')
ax.plot(x, data370, label='ssp370')
ax.plot(x, data585, label='ssp585')

# draw grid lines
plt.grid()

# add a title string
ax.set_title('CMIP6: tas global yearly mean')

# set axis labels
ax.set_xlabel('years')
ax.set_ylabel(data126.long_name+' ['+data126.units+']')

#add a legend
legend = ax.legend(loc='upper left', shadow=True, fontsize='small')
legend.get_frame().set_facecolor('lightgray')
```



1.7 Delete temporary files

```
[10]: cdo.cleanTempDir()
```

1.8 Xarray not CDO

And now let's see how much more effort we have to put in with xarray as opposed to CDO.

Load the packages xarray and numpy.

```
[11]: import xarray as xr
import numpy as np
```

Read the files for each scenario into an xarray.DataSet.

```
[12]: ds126 = xr.open_mfdataset(data_ssp126)
ds245 = xr.open_mfdataset(data_ssp245)
ds370 = xr.open_mfdataset(data_ssp370)
ds585 = xr.open_mfdataset(data_ssp585)
```

1.8.1 Yearly means

Compute the yearly means for each scenario.

```
[13]: tas126 = ds126.tas.resample(time='Y').mean()
      tas245 = ds245.tas.resample(time='Y').mean()
      tas370 = ds370.tas.resample(time='Y').mean()
      tas585 = ds585.tas.resample(time='Y').mean()
```

Since CDO uses the area weights for grid computations we have to generate them, because we want to compare the results later.

```
[14]: weights = np.cos(np.deg2rad(ds126.lat))
      weights.name = "weights"
```

Compute the data respecting the weights.

```
[15]: tas126_weighted = tas126.weighted(weights)
      tas245_weighted = tas245.weighted(weights)
      tas370_weighted = tas370.weighted(weights)
      tas585_weighted = tas585.weighted(weights)
```

Next step is to compute the field mean of the scenario data.

```
[16]: tas126_weightedmean = tas126_weighted.mean(['lon', 'lat'])
      tas245_weightedmean = tas245_weighted.mean(['lon', 'lat'])
      tas370_weightedmean = tas370_weighted.mean(['lon', 'lat'])
      tas585_weightedmean = tas585_weighted.mean(['lon', 'lat'])
```

1.9 Create the time series plot of scenarios

We do the same as above

1. define the plotting figure and axis
2. draw all 4 scenario time series
3. draw grid lines
4. add a title string
5. define the axis titles
6. add a legend

```
[17]: fig, ax = plt.subplots(figsize=(8,4))

      # draw scenario time series
      ax.plot(x, tas126_weightedmean, label='ssp126 weighted mean')
      ax.plot(x, tas245_weightedmean, label='ssp245 weighted mean')
      ax.plot(x, tas370_weightedmean, label='ssp370 weighted mean')
      ax.plot(x, tas585_weightedmean, label='ssp585 weighted mean')

      # add grid lines
      plt.grid()

      # add title string
```

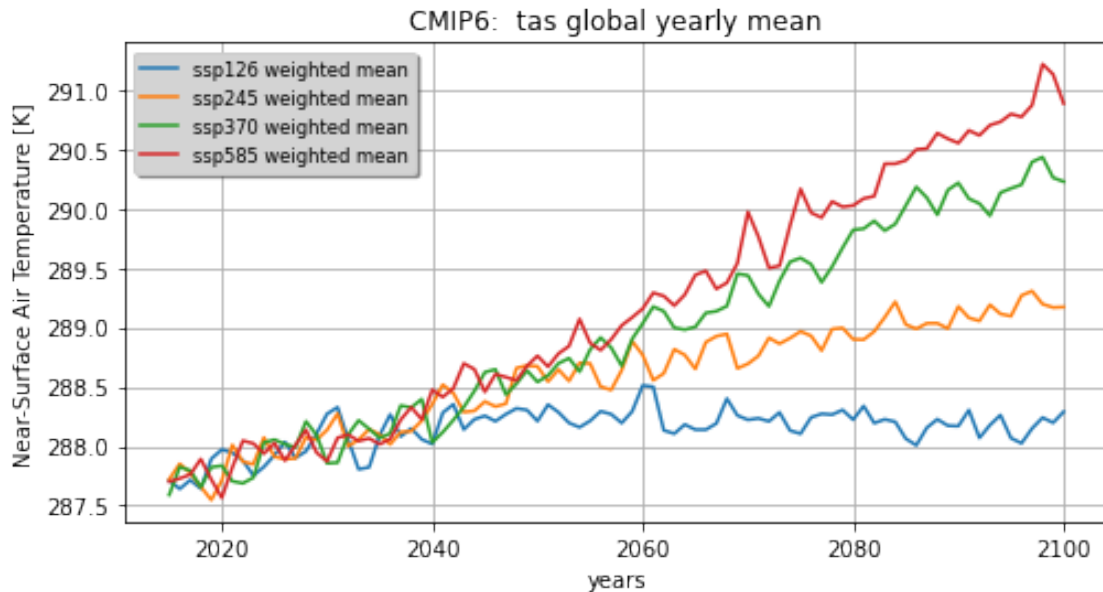
```

ax.set_title('CMIP6: tas global yearly mean')

# set axis labels
ax.set_xlabel('years')
ax.set_ylabel(data126.long_name+' [' +data126.units+']')

# add a legend
legend = ax.legend(loc='upper left', shadow=True, fontsize='small')
legend.get_frame().set_facecolor('lightgray')

```



2 Plot the historical and scenario time series together

In this case we have the problem that historical and scenario data have different time ranges. The solution is to define an x-axis for all years 1850 to 2100 and plot the data in their corresponding part of the time axis.

First, compute the historical annual mean data with CDO.

```

[18]: hist = '/pool/data/CMIP6/data/CMIP/MPI-M/MPI-ESM1-2-LR/historical/r1i1p1f1/Amon/
      ↪tas/gn/v20190710/tas_*.nc'

hist_mean = cdo.fldmean(options='-b F64 --reduce_dim ', input='-yearmean_
      ↪-mergetime ' + hist, returnXArray='tas')

```

Define a list variable x with the value range for all years.


```
[19]: x = range(1850,2101)
```

To plot the data accordingly to their time values we use array slicing. Therefore, for indexing we need the number of years in total and the number of years of the historical data.

```
[20]: xnum = len(x)          # number of all time steps
      xhnum = len(hist_mean) # number of time steps of historical
```

Now, we can create the plot with the time series of the historical and scenario data in one figure.

```
[21]: fig, ax = plt.subplots(figsize=(8,4))

      # draw empty plot
      plt.plot(xlim=(1850,2100), ylim=(280,300))

      # draw the time series
      ax.plot(x[0:xhnum], hist_mean, label='historical')
      ax.plot(x[165:xnum+1], data126, label='ssp126')
      ax.plot(x[165:xnum+1], data245, label='ssp245')
      ax.plot(x[165:xnum+1], data370, label='ssp370')
      ax.plot(x[165:xnum+1], data585, label='ssp585')

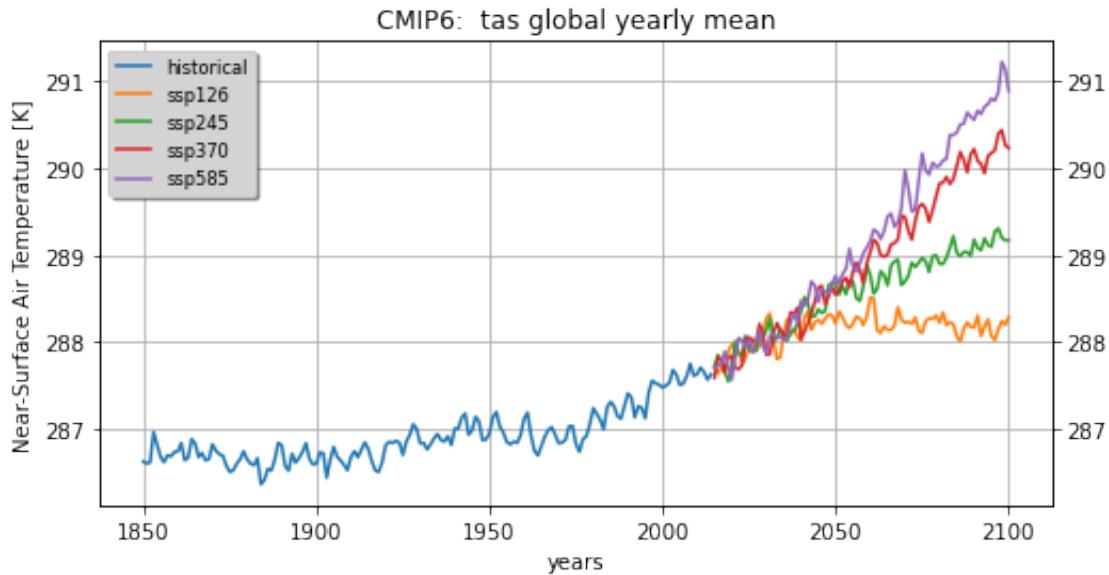
      # draw gridlines
      plt.grid()

      # set title
      ax.set_title('CMIP6: tas global yearly mean')

      # axis label settings
      ax.set_xlabel('years')
      ax.set_ylabel(data126.long_name+' [' +data126.units+']')
      ax.yaxis.set_ticks_position('both')
      ax.yaxis.set_label_position('right')
      ax.yaxis.set_label_position('left')

      # use y-axis annotations on both sides
      ax2 = ax.twinx()
      ax2.set_ylim(ax.get_ylim())

      # add a legend
      legend = ax.legend(loc='upper left', shadow=True, fontsize='small')
      legend.get_frame().set_facecolor('lightgray')
```



2.1 Create a plot for a poster with the seaborn library

We now repeat the same plot with the seaborn library. Choose settings that are suitable for using the plot in a poster. (Hint: set the seaborn style and context)

```
[22]: import seaborn as sns
```

For a poster presentation, any seaborn style with a white background will work. Here, we select `whitegrid`, you can also try `ticks` for a cleaner look. By setting the seaborn context to `poster`, the linewidth and axis label font sizes are automatically adjusted. They are large enough to read from a distance.

```
[23]: sns.set_style('whitegrid')
      sns.set_context('poster')
```

We use the same processed data as in the previous exercise.

We set the figure size to 10 inches width and 6.5 inches height. This should be suitable for an A0 poster. Then we use the seaborn `lineplot` function, note that it is required to specify *x* and *y* explicitly by passing them as arguments. Each line is annotated with a label, and that automatically creates a legend.

The labels for the x and y axis, as well as the plot title, are the same as in the previous exercise. To save space, we do not repeat the second y axis here.

```
[24]: fig, ax = plt.subplots(figsize=(10, 6.5))

      sns.lineplot(x=x[0:xhnum], y=hist_mean, label='historical')
```

```

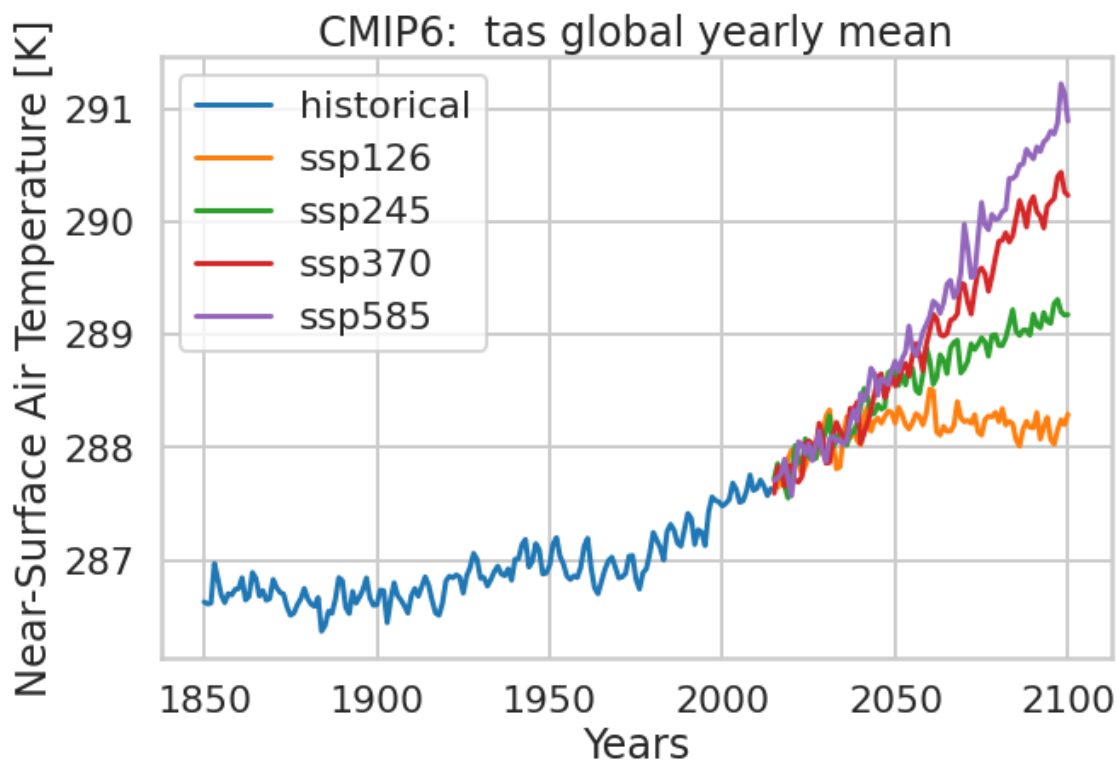
sns.lineplot(x=x[165:xnum+1], y=data126, label='ssp126')
sns.lineplot(x=x[165:xnum+1], y=data245, label='ssp245')
sns.lineplot(x=x[165:xnum+1], y=data370, label='ssp370')
sns.lineplot(x=x[165:xnum+1], y=data585, label='ssp585')

# axis label settings
ax.set_xlabel('Years')
ax.set_ylabel(data126.long_name+' [' +data126.units+']')

# set title
ax.set_title('CMIP6: tas global yearly mean')

plt.show()

```



[]: