IPCC AR6 Python tutorial

December 21, 2019

Mathias Hauser - Institute for Atmospheric and Climate Science, ETH Zurich, Zurich, Switzerland

1 Using the IPCC AR6 region definitions in Python

The Working Group I (WG 1) of the Sixt Assessment Report (AR6) of the Intergovernmental Panel on Climate Change (IPCC) defines new regions - in the following referred to as `AR6 regions`. This tutorial shows how the python package `regionmask` (Hauser, 2019) can be used to plot these regions, and how to create a mask for arbitrary latitude and longitude grids. We will use the package `xarray` (Hoyer and Hamman, 2017) to load netCDF files.

> Currently the AR6 regions are not included in a published version of `regionmask`, therefore it has to be directly installed from github.
>
> `pip install git+https://github.com/mathause/regionmask.git@feature/ar6_regions`
>
> The regions will be included in the next version (v0.6.0).

Import `regionmask` and check the version:

[1]:
```python
import regionmask
regionmask.__version__
```

[1]:
```
'0.5.0+dev'
```

Import `xarray` and check the version:

[2]:
```python
import xarray as xr
xr.__version__
```

[2]:
```
'0.14.1+32.g3cbc459c'
```

Import additional packages

[3]:
```python
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
import numpy as np
```

The regions are available at `regionmask.defined_regions.ar6`. Four variants of the regions are provided (`all`, `land`, `ocean`, and `separate_pacific`):

[4]:
```
regionmask.defined_regions.ar6
```

[4]:
```
Regions defined for the sixt IPCC assessment report

Attributes
----------
all : Regions
    All regions (land + ocean), regions split along the date line
    are combined (see below).
land : Regions
    Land regions only, regions split along the date line
    are combined (see below).
ocean : Regions
    Ocean regions only, regions split along the date line
    are combined (see below).
separate_pacific : Regions
    Original definitions of the regions, no combination of the pacific
    regions.

Combined Regions
----------------
SPO and SPO*; EPO and EPO*; NPO and NPO*

Note
----
The region numbers for all, land, and ocean are consistent. The
region numbers for all and separate_pacific are not.
```

We will illustrate the use of `regionmask` with `regionmask.defined_regions.ar6.all`

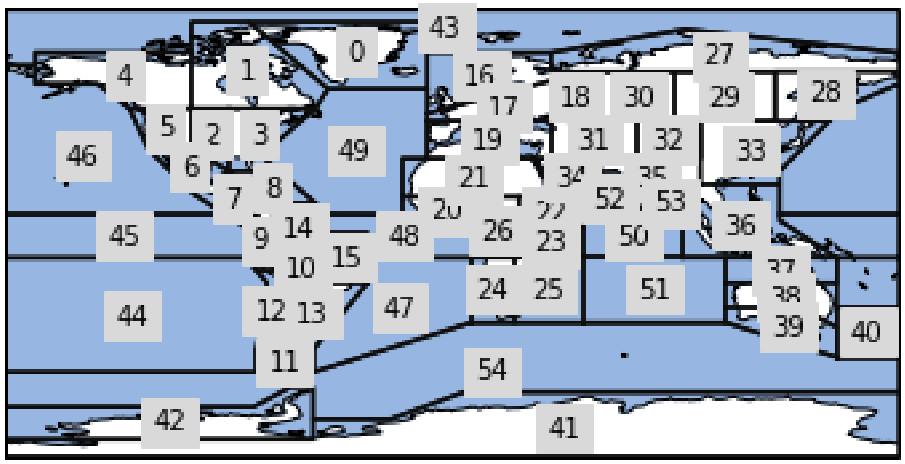
[5]:
```
ar6_all = regionmask.defined_regions.ar6.all

ar6_all
```

[5]:
```
55 'IPCC AR6 WGI Reference Regions (combined Pacific regions)' Regions
GIC NEC CNA ENA NWN WNA NCA SCA CAR NWS SAM SSA SWS SES NSA NES NEU CEU EEU MED
WAF SAH NEAF CEAF SWAF SEAF CAF RAR RFE ESB WSB WCA TIB EAS ARP SAS SEA NAU CAU
SAU NZ EAN WAN ARO SPO EPO NPO SAO EAO NAO EIO SIO ARS BOB SOO
```

1.1 Plotting

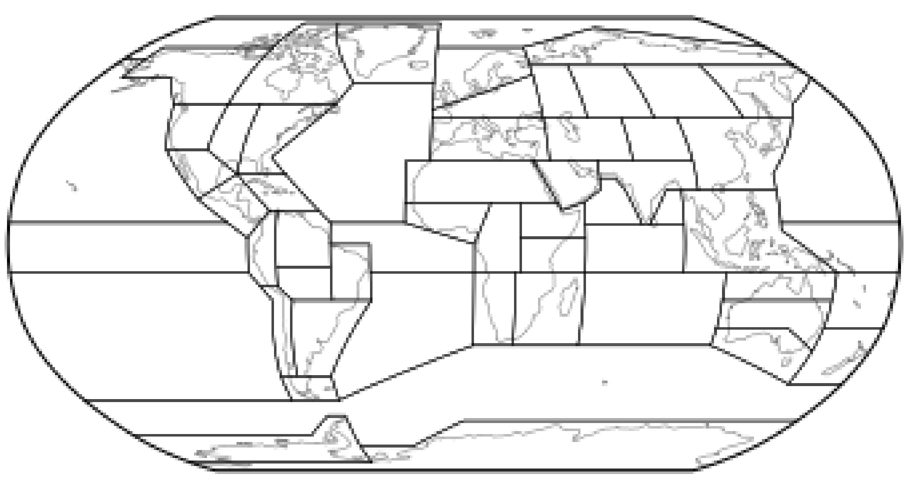
`ar6_all.plot()` creates an `cartopy` map plot including the outline of all regions. It returns a `axes` instance.

[6]:
```
ax = ar6_all.plot()
```



The plot can also be customized:

[7]:
```
f, ax = plt.subplots(subplot_kw=dict(projection=ccrs.Robinson()))

ax = ar6_all.plot(
    ax=ax,
    add_ocean=False,
    line_kws=dict(linewidth=0.5),
    coastlines=False,
    add_label=False,
);

ax.coastlines(color="0.5", lw=0.5);
```



1.2 Selecting Regions

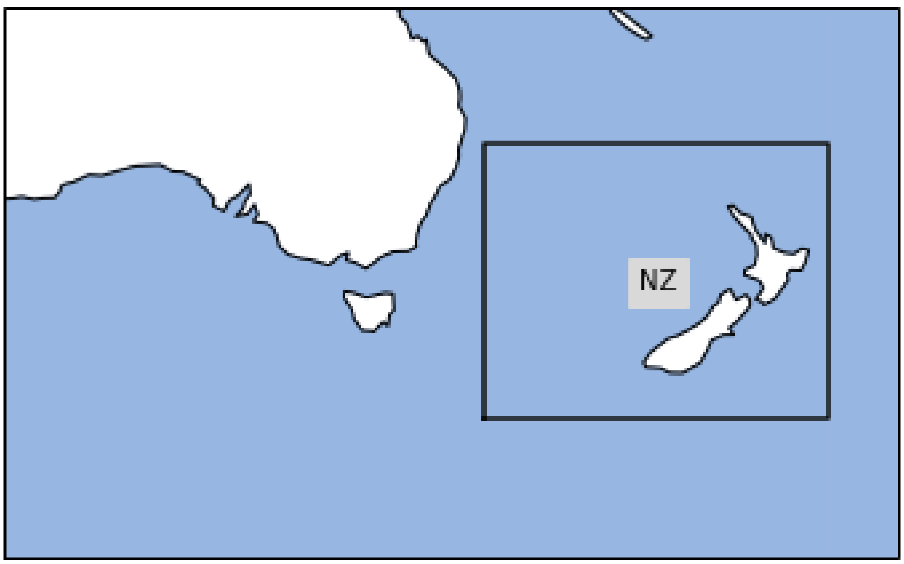
Select a single region (note the double brackets):

[8]:
```
ar6_all[[40]]
```

[8]:
```
1 'IPCC AR6 WGI Reference Regions (combined Pacific regions)' Regions
NZ
```

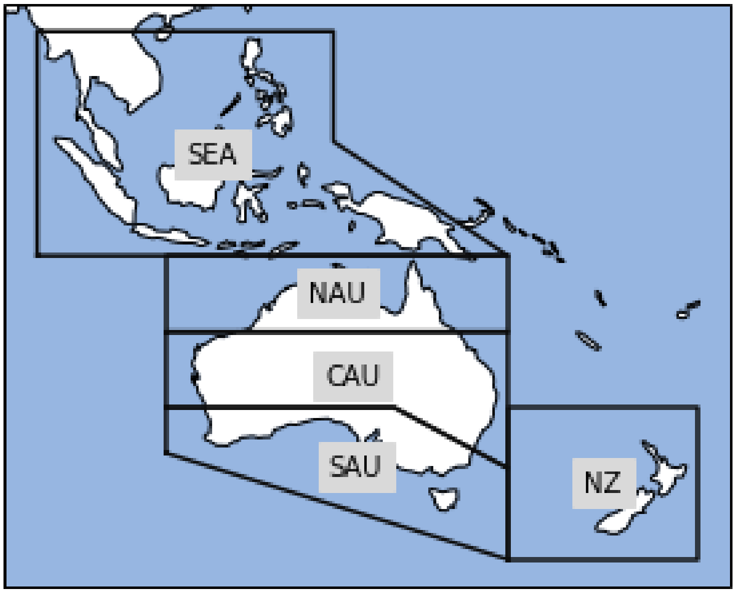
and plot it

[9]:
```
projection = ccrs.PlateCarree(central_longitude=180)

ax = ar6_all[[40]].plot(proj=projection, label="abbrev")

ax.set_extent([120, 185, -20, -60], ccrs.PlateCarree())
```



Regions can also be selected by their abbreviation, let's select several regions at once:

[10]:
```
australasia = ar6_all[["NZ", "SEA", "NAU", "CAU", "SAU"]]

ax = australasia.plot(proj=projection, label="abbrev")
```



1.3 Example Dataset

We load an example dataset using `xarray`.

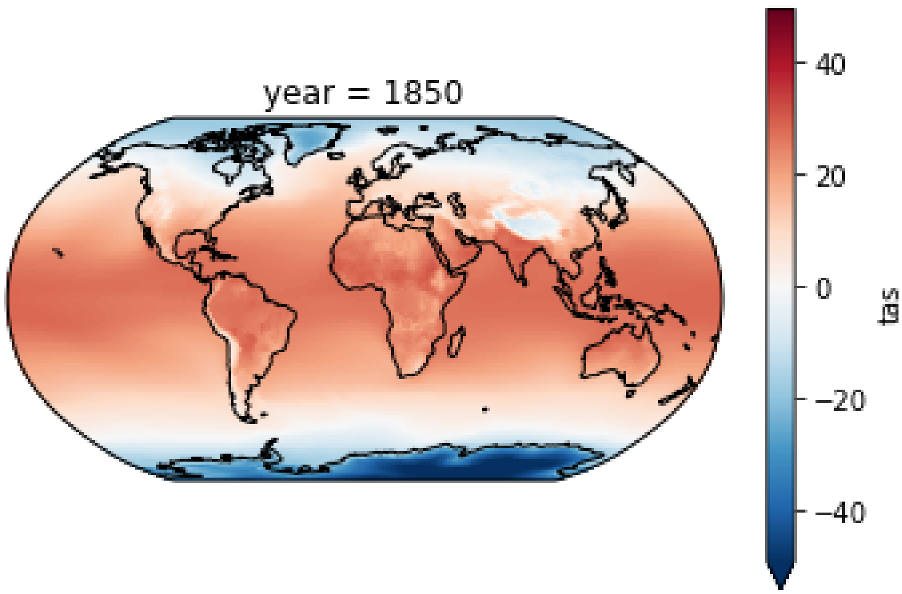
[11]:
```python
fN = ("/net/atmos/data/cmip6/historical/Amon/tas/CESM2/r1i1p1f1/gn/"
      "tas_Amon_CESM2_historical_r1i1p1f1_gn_185001-201412.nc")

ds = xr.open_dataset(fN)

# calculate annual mean
ds = ds.groupby("time.year").mean("time")

# extract tas and convert to celsius
tas = ds.tas - 273.15
```

```
/net/exo/landclim/mathause/.conda/envs/regionmask-docs/lib/python3.7/site-
packages/xarray/conventions.py:494: SerializationWarning: variable 'tas' has
multiple fill values {1e+20, 1e+20}, decoding all values to NaN.
  use_cftime=use_cftime,
```

The example data is a temperature field. Let's plot the first time step:

[12]:
```python
proj = ccrs.Robinson()

f, ax = plt.subplots(subplot_kw=dict(projection=proj))

tas.isel(year=0).plot.pcolormesh(
```

```
    ax=ax, transform=ccrs.PlateCarree(), robust=True, center=0
)

ax.coastlines();
```



1.4 Creating a region mask

Using `ar6_all.mask(lon, lat)` we can create an array where each grid cell is numbered according to the region it belongs to. Grid cells that are not part of any region are `NaN`. We can directly pass an `xarray` object containing `lon` and `lat` coordinates to the `mask` function. Let's only use the land regions for this:

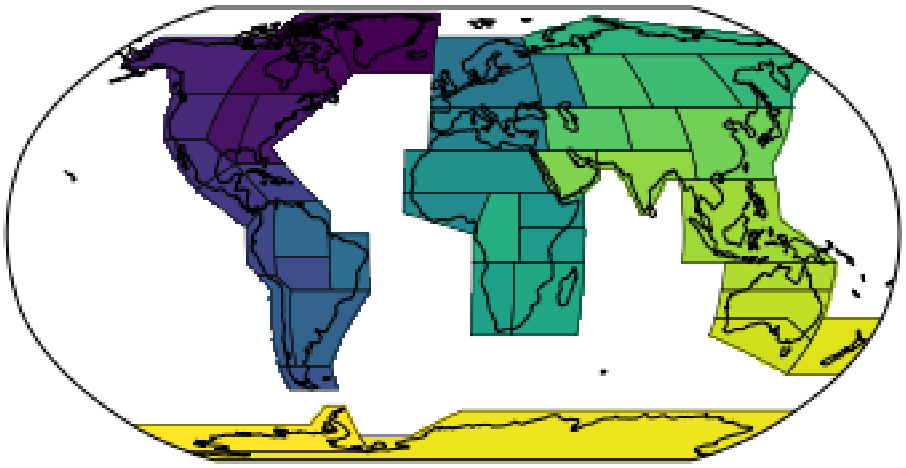
[13]:
```
ar6_land = regionmask.defined_regions.ar6.land
```

[14]:
```
mask = ar6_land.mask(tas)
```

Let's plot the (region) `mask`:

[15]:
```
proj = ccrs.Robinson()
f, ax = plt.subplots(subplot_kw=dict(projection=proj))

h = mask.plot.pcolormesh(
    ax=ax, transform=ccrs.PlateCarree(), add_colorbar=False
)

ax.coastlines()
```

```
ar6_land.plot_regions(line_kws=dict(lw=0.5), add_label=False);
```



We want to select the region "Central North America" (CNA). Thus we first need to find out which number the region has:

[16]:
```
CNA = ar6_land.map_keys("CNA")

CNA
```

[16]: 2

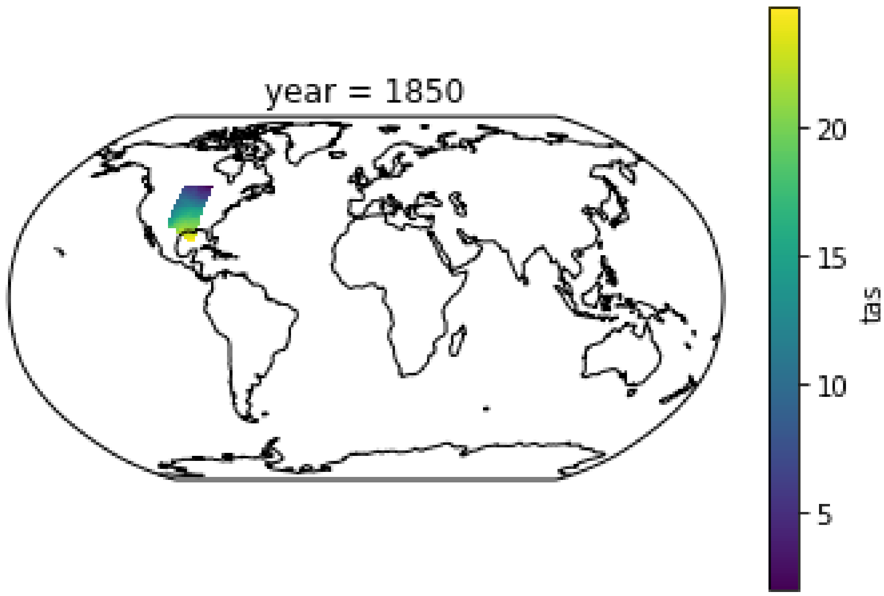
1.5 Select using `where`

`xarray` provides the `where` function which assigns `NaN` values to all grid points that are `False`:

[17]:
```
tas_CNA = tas.where(mask == CNA)
```

Check everything went well by repeating the first plot with the selected region:

[18]:
```
proj = ccrs.Robinson()

ax = plt.subplot(111, projection=proj)

tas_CNA.isel(year=0).plot.pcolormesh(ax=ax, transform=ccrs.PlateCarree())

ax.coastlines();
```



Looks good - let's take the area average and plot the time series. Each grid cell should be weighted by its correspoinding area. For the rectangular grid used here the cosine of the latitude is a good approximation. Unfortunately a weighted mean is not yet (as of version 0.14) implemented in xarray. Therefore, we have to define our own function (see pydata/xarray/#2922 for details).

[19]:
```
def weighted_mean(da, weights, dim):
    """Reduce da by a weighted mean along some dimension(s).

    Parameters
    ----------
    da : DataArray
        Object over which the weighted reduction operation is applied.
    weights : DataArray
        An array of weights associated with the values in this Dataset.
    dim : str or sequence of str, optional
        Dimension(s) over which to apply the weighted `mean`.

    Returns
    -------
    weighted_mean : DataArray
        New DataArray with weighted mean applied to its data and
        the indicated dimension(s) removed.
    """

    weighted_sum = (da * weights).sum(dim=dim, skipna=True)
    # need to mask weights where data is not valid
    masked_weights = weights.where(da.notnull())
```

```
    sum_of_weights = masked_weights.sum(dim=dim, skipna=True)
    valid_weights = sum_of_weights != 0
    sum_of_weights = sum_of_weights.where(valid_weights)

    return weighted_sum / sum_of_weights
```

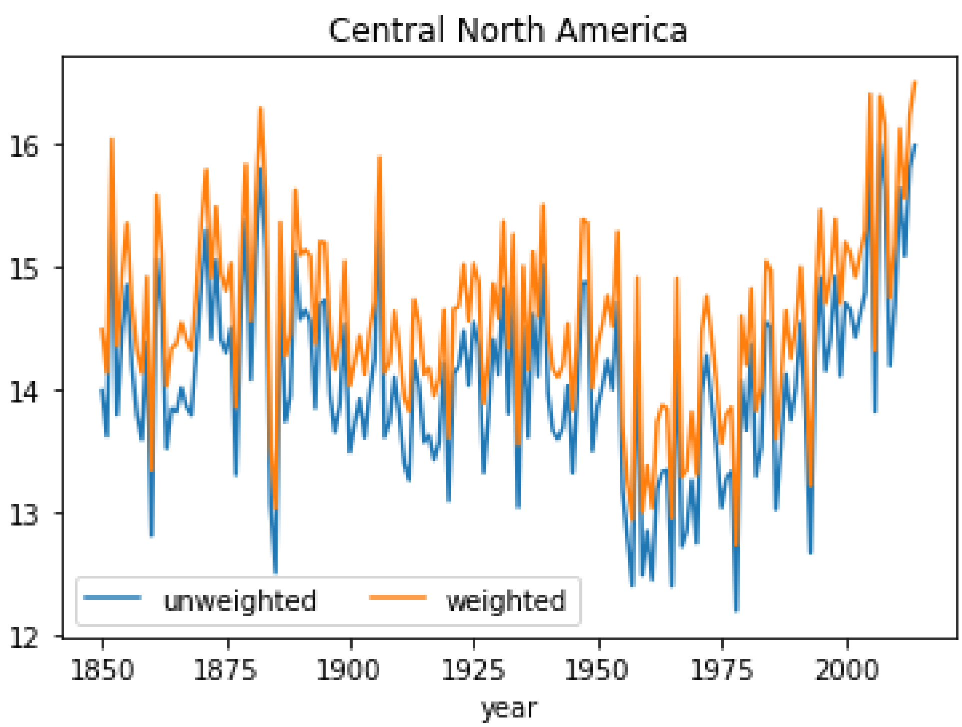
Let's compare the weighed with the unweighted mean it that actually makes a difference.

[20]:
```
weights = np.cos(np.deg2rad(tas.lat))

ts_tas_CNA_unweighted = tas_CNA.mean(('lat', 'lon'))
ts_tas_CNA_weighted = weighted_mean(tas_CNA, weights, ('lat', 'lon'))
```

add the line plot

[21]:
```
f, ax = plt.subplots()
ts_tas_CNA_unweighted.plot(ax=ax, label='unweighted')
ts_tas_CNA_weighted.plot(ax=ax, label='weighted')

plt.legend(ncol=2);
ax.set_title("Central North America")
```

[21]: Text(0.5, 1.0, 'Central North America')



Note how the weighted mean is larger than the unweighted..

1.6 Using regionmask in production

It is not very practical to calculate each region mean by hand, therefore a function is provided that calculates the weighted average for all regions and additionaly adds the global mean, ocean mean, land mean, and a mean over all land except Antarctica,

[22]:
```
def region_average(da, regions, land_only=True):
    """ Calculate regional average

        Parameters
        ----------
        da : DataArray
            Object over which the weighted reduction operation is applied.
        regions : regionmask.Regions
            regions to take the average over.
        land_only : bool, optional
            Whether to mask out ocean points before calculating regional
            means.

        Returns
        -------
        reg_ave : DataArray
            New DataArray with averaged over the whole globe, the ocean,
            the land, the land without Antarctica, and all regions.
            Dimensions (n_regions + 4) x (time)
    """

    if not isinstance(regions, regionmask.Regions):
        raise ValueError("'regions' must be a regionmask.Regions instance")

    abbrevs = ["global", "ocean", "land", "land_wo_antarctica"]
    abbrevs = abbrevs + regions.abbrevs

    numbers = np.array(regions.numbers)

    # get cosine weights
    weight = np.cos(np.deg2rad(da.lat))

    # get a land mask
    landmask = regionmask.defined_regions.natural_earth.land_110.mask(da)
    landmask = landmask == 0

    if land_only:
        wgt = weight * landmask
    else:
```

```
    # we need to add lon coordinates
    wgt = xr.full_like(landmask, 1) * weight

mask = regions.mask(da)

# list to accumulate averages
ave = list()

# global mean
a = weighted_mean(da, dim=("lat", "lon"), weights=weight)
ave.append(a)

# global ocean mean
weights = (weight * (1.0 - landmask))
a = weighted_mean(da, dim=("lat", "lon"), weights=weights)
ave.append(a)

# global land mean
weights = (weight * landmask)
a = weighted_mean(da, dim=("lat", "lon"), weights=weights)
ave.append(a)

# global land mean w/o antarctica
da_selected = da.sel(lat=slice(-60, None))
weights = (weight * landmask)
a = weighted_mean(da_selected, dim=("lat", "lon"), weights=weights)
ave.append(a)

# it is faster to calculate the weighted mean via groupby
g = da.groupby(mask)
wgt_stacked = wgt.stack(stacked_lat_lon=("lat", "lon"))
a = g.apply(weighted_mean, dim=("stacked_lat_lon"), weights=wgt_stacked)
ave.append(a.drop("region"))

da = xr.concat(ave, dim="region")

# shift region coordinates such that the numbers correspond to the
# regions
numbers = np.arange(numbers.min() - 4, numbers.max() + 1)

# add the abbreviations of the regions, update the numbers
da = da.assign_coords(
    **{"abbrev": ("region", abbrevs), "number": ("region", numbers)}
)

# create a multiindex
da = da.set_index(region=("abbrev", "number"))
```

```python
    return da
```

We set `land_only=False` so that the result is comparable to `ts_tas_CNA_weighted`. The function can be used like so:

[23]:
```python
tas_reg_ave = region_average(tas, ar6_land, land_only=False)
```

It returns a `DataArray` of the form `n_regions + 4` x `time`.

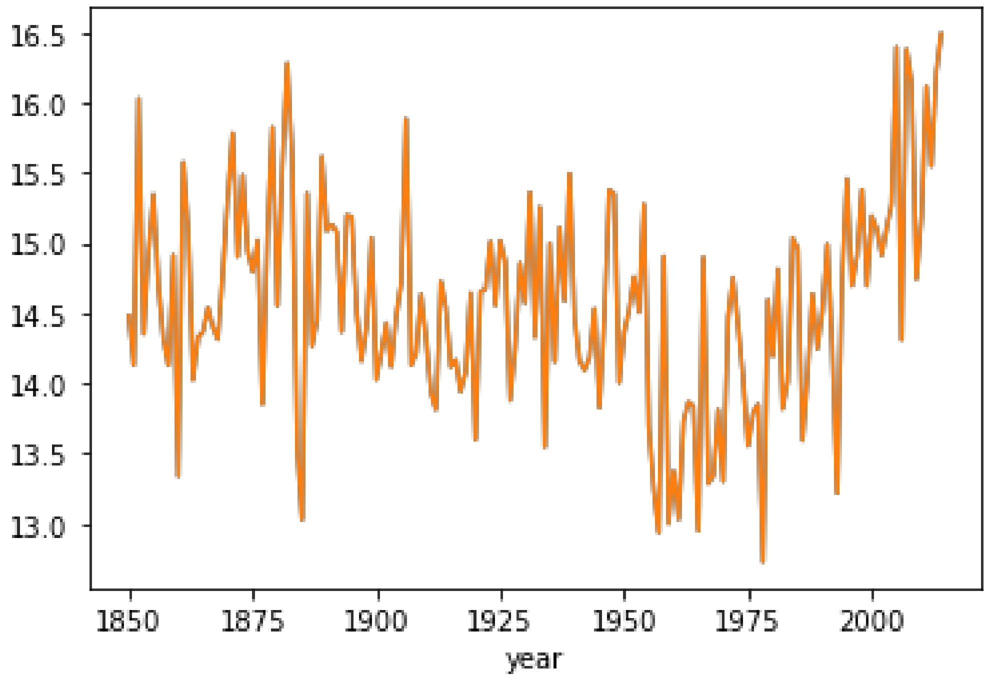
[24]:
```python
print(tas_reg_ave)
```

```
<xarray.DataArray (year: 165, region: 47)>
array([[ 14.05570509,  15.94685318,   9.40013865, …,  13.73055188,
        -31.64397203, -19.63909115],
       [ 14.14698735,  15.94150805,   9.72929508, …,  13.83234177,
        -30.65394993, -19.60219786],
       [ 13.98464357,  15.80148087,   9.51201302, …,  14.23370685,
        -31.23482196, -20.11581554],
       …,
       [ 15.23616381,  16.9465906 ,  11.02549118, …,  14.43705979,
        -29.60729028, -16.8629853 ],
       [ 15.15337885,  16.88072324,  10.90105902, …,  14.96161691,
        -30.37215337, -17.7697203 ],
       [ 14.9954769 ,  16.72866722,  10.72876572, …,  15.34736543,
        -29.6835183 , -17.97165143]])
Coordinates:
  * year     (year) int64 1850 1851 1852 1853 1854 … 2010 2011 2012 2013 2014
  * region   (region) MultiIndex
  - abbrev   (region) object 'global' 'ocean' 'land' … 'NZ' 'EAN' 'WAN'
  - number   (region) int64 -4 -3 -2 -1 0 1 2 3 4 … 34 35 36 37 38 39 40 41 42
```

Regions can be selected by abbreviation (`tas_reg_ave.sel(abbrev="CNA")`) or number (`tas_reg_ave.sel(number=2)`). Compare the mean over `CNA` calculated with both methods:

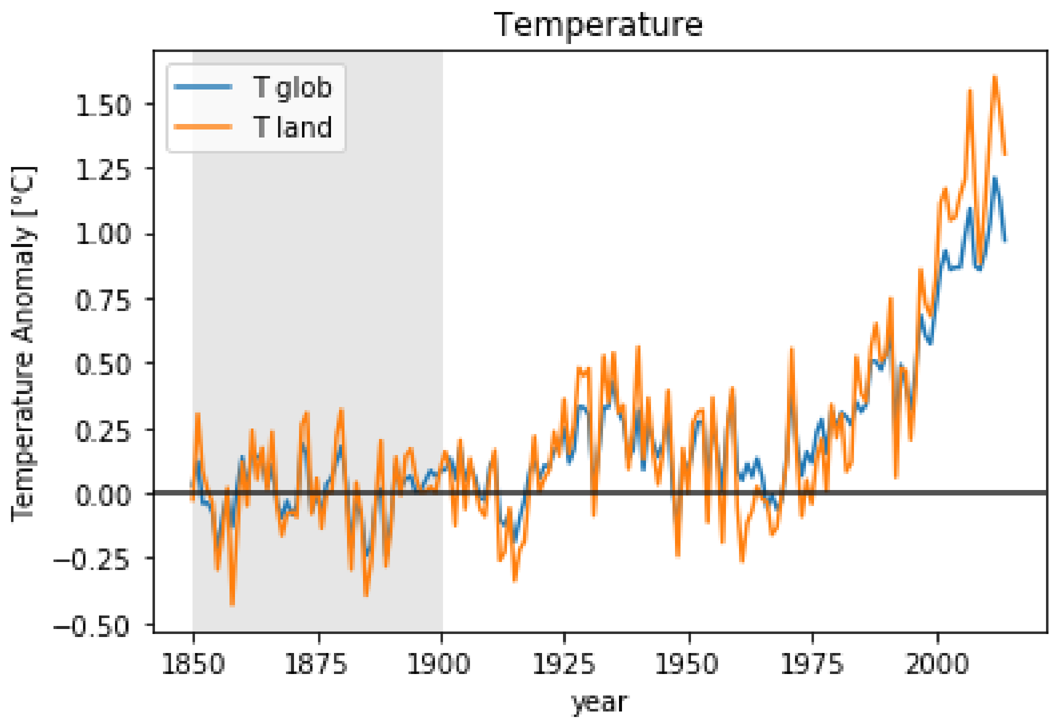
[25]:
```python
tas_reg_ave.sel(abbrev="CNA").plot()

ts_tas_CNA_weighted.plot.line()
```

[25]: `[<matplotlib.lines.Line2D at 0x2afc4795f860>]`



The match - as a last thing we compare the global mean temperature with the land mean temperature:

[26]:
```
glob = tas_reg_ave.sel(abbrev="global")
land = tas_reg_ave.sel(abbrev="land")

# calculate anomalies wrt 1850 - 1900
glob -= glob.sel(year=slice(1850, 1900)).mean()
land -= land.sel(year=slice(1850, 1900)).mean()

# ==

f, ax = plt.subplots()

glob.plot(ax=ax, label="T glob")
land.plot(ax=ax, label="T land")


ax.legend(loc="upper left")

ax.axvspan(1850, 1900, color="0.9")
ax.axhline(0, color="0.1")

ax.set_ylabel("Temperature Anomaly [°C]")
ax.set_title("Temperature");
```



1.7 References

Hauser M (2019), Regionmask: plotting and creation of masks of spatial regions in Python, *Zenodo*, doi:10.5281/zenodo.3585543.

Hoyer, S. and Hamman, J., 2017. xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*, 5(1), p.10. DOI: doi:10.5334/jors.148.