python_cdo_introduction

August 10, 2022

1 CDO - Climate Data Operators

Note: This notebook comes from the ***DKRZ Python Course for Geoscientists*** and has been slightly expanded.

The scripting language package **python-cdo** is essentially a wrapper around the CDO binary. It parses method arguments and options, builds a command line and executes it. There is no shared library backend which calls CDO operators. This has some advantages:

1. operator chaining is fully supported
2. multiple CDO binaries can be used at the same time using setCdo()
3. packages are highly portable, because they are pure python implementations

1.0.1 CDO documentation

CDO User Guide https://code.mpimet.mpg.de/projects/cdo/embedded/cdo.pdf

CDO Python bindings introduction https://code.mpimet.mpg.de/attachments/download/18824/cdo-bindings.pdf

1.0.2 Installation

Install python-cdo with conda:

```
conda install -c conda-forge python-cdo
```

1.0.3 Table of contents

- Show information about the variables value range, times, levels in more detail
- Copy file
- Select variables
- Select timesteps
- Operator chaining
- Select a sub-region
- Compute the field mean
- Assign data to variable
- Remapping
- Remap data to grid of another file
- Create a land-sea mask
- Masking data
- Delete temporary files

1.1 About CDO

See also https://code.mpimet.mpg.de/projects/cdo/wiki

CDO is a large tool set for working on climate and NWP model data. NetCDF 3/4, GRIB 1/2 including SZIP (or AEC) and JPEG compression, EXTRA, SERVICE and IEG are supported as IO-formats. Apart from that CDO can be used to analyse any kind of gridded data not related to climate science. CDO has very small memory requirements and can process files larger than the physical memory.

CDO is open source and released under the terms of the GNU General Public License v2 (GPL).

1.2 Import CDO module

Import the CDO module and set cdo to Cdo() which makes writing a little easier.

[1]:
```
from cdo import *

cdo = Cdo()
```

1.3 Show CDO version

python-cdo version:

[2]:
```
print(cdo.__version__())
```

```
1.5.4
```

Based on CDO version:

[3]:
```
print(cdo.version())
```

```
2.0.5
```

1.4 Set temporary directory

Set another directory for storing tempfiles with a constructor option and remove anything left in there when you experienced a crash or something like this

[4]:
```
tempPath = './tmp/'
cdo = Cdo(tempdir=tempPath)
```

1.5 Delete temporary files

[5]:
```
cdo.cleanTempDir()
```

1.6 List CDO operators

More than 800 operators are available. To display the list of all operators you can use **operators**.

[6]:
```
#cdo.operators
```

1.7 List all operators starting with 'sel'

Use list comprehension

[7]:
```
[key for key, value in cdo.operators.items() if key.startswith('sel')]
```

[7]:
```
['selall',
 'selcircle',
 'selcode',
 'seldate',
 'selday',
 'select',
 'selgrid',
 'selgridcell',
 'selgridname',
 'selhour',
 'selindex',
 'selindexbox',
 'sellevel',
 'sellevidx',
 'sellonlatbox',
 'selltype',
 'selmon',
 'selmonth',
```

```
 'selmulti',
 'selname',
 'seloperator',
 'selparam',
 'selrec',
 'selregion',
 'selseas',
 'selseason',
 'selsmon',
 'selstdname',
 'seltabnum',
 'seltime',
 'seltimeidx',
 'seltimestep',
 'selvar',
 'selyear',
 'selyearidx',
 'selzaxis',
 'selzaxisname']
```

1.8 Turn debugging on/off

Use the debug method of CDO to turn on or off debugging.

[8]:
```
cdo.debug = True

cdo.debug = False
```

1.9 Show information about file content

[9]:
```
cdo.sinfon(input='/Users/k204045/data/rectilinear_grid_2D.nc')
```

[9]:
```
['File format : NetCDF',
 '-1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter
name',
 '1 : unknown  unknown  v instant       1   1     18432   1  F32  : tsurf',
 '2 : unknown  unknown  v instant       1   1     18432   1  F32  : precip',
 '3 : unknown  unknown  v instant       1   1     18432   1  F32  : u10',
 '4 : unknown  unknown  v instant       1   1     18432   1  F32  : v10',
 '5 : unknown  unknown  v instant       1   1     18432   1  F32  : qvi',
 '6 : unknown  unknown  v instant       1   1     18432   1  F32  : slp',
 'Grid coordinates :',
 '1 : gaussian                 : points=18432 (192x96)',
 'lon : -180 to 178.125 by 1.875 degrees_east  circular',
 'lat : 88.57217 to -88.57217 degrees_north',
```

```
 'Vertical coordinates :',
 '1 : surface                   : levels=1',
 'Time coordinate :',
 'time : 40 steps',
 'RefTime =  2001-01-01 00:00:00  Units = hours  Calendar = standard',
 'YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD
hh:mm:ss',
 '2001-01-01 00:00:00  2001-01-01 06:00:00  2001-01-01 12:00:00  2001-01-01
18:00:00',
 '2001-01-02 00:00:00  2001-01-02 06:00:00  2001-01-02 12:00:00  2001-01-02
18:00:00',
 '2001-01-03 00:00:00  2001-01-03 06:00:00  2001-01-03 12:00:00  2001-01-03
18:00:00',
 '2001-01-04 00:00:00  2001-01-04 06:00:00  2001-01-04 12:00:00  2001-01-04
18:00:00',
 '2001-01-05 00:00:00  2001-01-05 06:00:00  2001-01-05 12:00:00  2001-01-05
18:00:00',
 '2001-01-06 00:00:00  2001-01-06 06:00:00  2001-01-06 12:00:00  2001-01-06
18:00:00',
 '2001-01-07 00:00:00  2001-01-07 06:00:00  2001-01-07 12:00:00  2001-01-07
18:00:00',
 '2001-01-08 00:00:00  2001-01-08 06:00:00  2001-01-08 12:00:00  2001-01-08
18:00:00',
 '2001-01-09 00:00:00  2001-01-09 06:00:00  2001-01-09 12:00:00  2001-01-09
18:00:00',
 '2001-01-10 00:00:00  2001-01-10 06:00:00  2001-01-10 12:00:00  2001-01-10
18:00:00']
```

1.10 Display information about the file content

[10]:
```
infile = '/Users/k204045/data/rectilinear_grid_2D.nc'

#cdo.sinfon(input=infile)
```

In comparison to xarray's dataset information.

[11]:
```
import xarray as xr

ds = xr.open_dataset('/Users/k204045/data/rectilinear_grid_2D.nc')

print(ds.info())
```

```
xarray.Dataset {
dimensions:
        lon = 192 ;
        lat = 96 ;
```

```
        time = 40 ;

variables:
        float64 lon(lon) ;
                lon:standard_name = longitude ;
                lon:long_name = longitude ;
                lon:units = degrees_east ;
                lon:axis = X ;
        float64 lat(lat) ;
                lat:standard_name = latitude ;
                lat:long_name = latitude ;
                lat:units = degrees_north ;
                lat:axis = Y ;
        datetime64[ns] time(time) ;
                time:standard_name = time ;
        float32 tsurf(time, lat, lon) ;
                tsurf:long_name = surface temperature ;
                tsurf:units = K ;
                tsurf:code = 169 ;
                tsurf:table = 128 ;
                tsurf:grid_type = gaussian ;
        float32 precip(time, lat, lon) ;
                precip:long_name = total precipitation ;
                precip:units = kg/m^2s ;
                precip:code = 4 ;
                precip:table = 128 ;
                precip:grid_type = gaussian ;
        float32 u10(time, lat, lon) ;
                u10:long_name = 10m u-velocity ;
                u10:units = m/s ;
                u10:code = 165 ;
                u10:table = 128 ;
                u10:grid_type = gaussian ;
        float32 v10(time, lat, lon) ;
                v10:long_name = 10m v-velocity ;
                v10:units = m/s ;
                v10:code = 166 ;
                v10:table = 128 ;
                v10:grid_type = gaussian ;
        float32 qvi(time, lat, lon) ;
                qvi:long_name = vertically integrated water vapor ;
                qvi:units = kg/m^2 ;
                qvi:code = 230 ;
                qvi:table = 128 ;
                qvi:grid_type = gaussian ;
        float32 slp(time, lat, lon) ;
                slp:long_name = mean sea level pressure ;
                slp:units = Pa ;
```

```
                slp:code = 151 ;
                slp:table = 128 ;
                slp:grid_type = gaussian ;

// global attributes:
}None
```

1.11 Show information about the variables value range, times, levels in more detail

[12]:
```
#cdo.infon(input='/Users/k204045/data/rectilinear_grid_2D.nc')
```

1.12 Copy file

Copy the input file to outfile.nc and change the data precision from float32 to float64.

[13]:
```
cdo.copy(input='/Users/k204045/data/rectilinear_grid_2D.nc', options='-b F64',␣
 ↪output='outfile.nc')
cdo.sinfon(input='outfile.nc')
```

[13]:
```
['File format : NetCDF',
 '-1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter
name',
 '1 : unknown  unknown  v instant       1   1     18432   1  F64  : tsurf',
 '2 : unknown  unknown  v instant       1   1     18432   1  F64  : precip',
 '3 : unknown  unknown  v instant       1   1     18432   1  F64  : u10',
 '4 : unknown  unknown  v instant       1   1     18432   1  F64  : v10',
 '5 : unknown  unknown  v instant       1   1     18432   1  F64  : qvi',
 '6 : unknown  unknown  v instant       1   1     18432   1  F64  : slp',
 'Grid coordinates :',
 '1 : gaussian                  : points=18432 (192x96)',
 'lon : -180 to 178.125 by 1.875 degrees_east  circular',
 'lat : 88.57217 to -88.57217 degrees_north',
 'Vertical coordinates :',
 '1 : surface                   : levels=1',
 'Time coordinate :',
 'time : 40 steps',
 'RefTime =  2001-01-01 00:00:00  Units = hours  Calendar = standard',
 'YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD
hh:mm:ss',
 '2001-01-01 00:00:00  2001-01-01 06:00:00  2001-01-01 12:00:00  2001-01-01
18:00:00',
 '2001-01-02 00:00:00  2001-01-02 06:00:00  2001-01-02 12:00:00  2001-01-02
18:00:00',
 '2001-01-03 00:00:00  2001-01-03 06:00:00  2001-01-03 12:00:00  2001-01-03
18:00:00',
```

```
 '2001-01-04 00:00:00  2001-01-04 06:00:00  2001-01-04 12:00:00  2001-01-04
18:00:00',
 '2001-01-05 00:00:00  2001-01-05 06:00:00  2001-01-05 12:00:00  2001-01-05
18:00:00',
 '2001-01-06 00:00:00  2001-01-06 06:00:00  2001-01-06 12:00:00  2001-01-06
18:00:00',
 '2001-01-07 00:00:00  2001-01-07 06:00:00  2001-01-07 12:00:00  2001-01-07
18:00:00',
 '2001-01-08 00:00:00  2001-01-08 06:00:00  2001-01-08 12:00:00  2001-01-08
18:00:00',
 '2001-01-09 00:00:00  2001-01-09 06:00:00  2001-01-09 12:00:00  2001-01-09
18:00:00',
 '2001-01-10 00:00:00  2001-01-10 06:00:00  2001-01-10 12:00:00  2001-01-10
18:00:00']
```

1.13 Output to file or assign to variable?

You can write the result to an output file or assign it to a Python variable as a Numpy/Xarray Array or as an Xarray dataset.

Variable assignments:

```
returnMaArray
returnXArray
returnXDataset
```

To make things easier for us, we define the variables infile and outfile for the input and output files.

[14]:
```
infile  = '/Users/k204045/data/rectilinear_grid_2D.nc'
outfile = './outfile.nc'
```

Save result to output file If the results are only to be used in a tempo-dependent manner, the file output can be left to CDO. For this the parameter **output** is not used this time, but the return value of the CDO call (filename, unique, temporary) is assigned to a variable.

1. with a given output file name

[15]:
```
cdo.selvar('tsurf', input=infile, output=outfile)
```

[15]:
```
'./outfile.nc'
```

2. without explicit output file name

[16]:
```
ts_filename = cdo.selvar('tsurf', input=infile)
ts_filename
```

[16]:
```
'/Users/k204045/Python/CDO/tmp/cdoPyz9bgduht'
```

Assign the result to a Python variable

[17]:
```
ts_ma = cdo.selvar('tsurf', input=infile, returnMaArray='tsurf')
print(type(ts_ma))
#print(ts_ma)
```

```
<class 'numpy.ma.core.MaskedArray'>
```

[18]:
```
ts_xa = cdo.selvar('tsurf', input=infile, returnXArray='tsurf')
print(ts_xa)
```

```
<xarray.DataArray 'tsurf' (time: 40, lat: 96, lon: 192)>
[737280 values with dtype=float32]
Coordinates:
  * time     (time) datetime64[ns] 2001-01-01 … 2001-01-10T18:00:00
  * lon      (lon) float64 -180.0 -178.1 -176.2 -174.4 … 174.4 176.2 178.1
  * lat      (lat) float64 88.57 86.72 84.86 83.0 … -83.0 -84.86 -86.72 -88.57
Attributes:
    long_name:      surface temperature
    units:          K
    code:           169
    table:          128
    CDI_grid_type:  gaussian
```

[19]:
```
ts_ds = cdo.selvar('tsurf', input=infile, returnXDataset='tsurf')
print(ts_ds)
```

```
<xarray.Dataset>
Dimensions:  (time: 40, lon: 192, lat: 96)
Coordinates:
  * time     (time) datetime64[ns] 2001-01-01 … 2001-01-10T18:00:00
  * lon      (lon) float64 -180.0 -178.1 -176.2 -174.4 … 174.4 176.2 178.1
  * lat      (lat) float64 88.57 86.72 84.86 83.0 … -83.0 -84.86 -86.72 -88.57
Data variables:
    tsurf    (time, lat, lon) float32 …
Attributes:
    CDI:          Climate Data Interface version 2.0.5 (https://mpimet.mpg.de…
    Conventions:  CF-1.6
    history:      Wed Aug 10 15:16:58 2022: cdo -O -s -f nc -selvar,tsurf /Us…
    CDO:          Climate Data Operators version 2.0.5 (https://mpimet.mpg.de…
```

1.14 Select variables

[20]:
```
cdo.selvar('tsurf', input=infile, output=outfile)
cdo.sinfon(input=outfile)
```

[20]:
```
['File format : NetCDF',
 '-1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter
name',
 '1 : unknown  unknown  v instant       1   1     18432   1  F32  : tsurf',
 'Grid coordinates :',
 '1 : gaussian                 : points=18432 (192x96)',
 'lon : -180 to 178.125 by 1.875 degrees_east  circular',
 'lat : 88.57217 to -88.57217 degrees_north',
 'Vertical coordinates :',
 '1 : surface                  : levels=1',
 'Time coordinate :',
 'time : 40 steps',
 'RefTime =  2001-01-01 00:00:00  Units = hours  Calendar = standard',
 'YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD
hh:mm:ss',
 '2001-01-01 00:00:00  2001-01-01 06:00:00  2001-01-01 12:00:00  2001-01-01
18:00:00',
 '2001-01-02 00:00:00  2001-01-02 06:00:00  2001-01-02 12:00:00  2001-01-02
18:00:00',
 '2001-01-03 00:00:00  2001-01-03 06:00:00  2001-01-03 12:00:00  2001-01-03
18:00:00',
 '2001-01-04 00:00:00  2001-01-04 06:00:00  2001-01-04 12:00:00  2001-01-04
18:00:00',
 '2001-01-05 00:00:00  2001-01-05 06:00:00  2001-01-05 12:00:00  2001-01-05
18:00:00',
 '2001-01-06 00:00:00  2001-01-06 06:00:00  2001-01-06 12:00:00  2001-01-06
18:00:00',
 '2001-01-07 00:00:00  2001-01-07 06:00:00  2001-01-07 12:00:00  2001-01-07
18:00:00',
 '2001-01-08 00:00:00  2001-01-08 06:00:00  2001-01-08 12:00:00  2001-01-08
18:00:00',
 '2001-01-09 00:00:00  2001-01-09 06:00:00  2001-01-09 12:00:00  2001-01-09
18:00:00',
 '2001-01-10 00:00:00  2001-01-10 06:00:00  2001-01-10 12:00:00  2001-01-10
18:00:00']
```

[21]:
```
cdo.selvar('u10,v10', input=infile, output=outfile)
cdo.sinfon(input=outfile)
```

[21]:
```
['File format : NetCDF',
 '-1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter
name',
 '1 : unknown  unknown  v instant       1   1     18432   1  F32  : u10',
 '2 : unknown  unknown  v instant       1   1     18432   1  F32  : v10',
 'Grid coordinates :',
 '1 : gaussian                 : points=18432 (192x96)',
 'lon : -180 to 178.125 by 1.875 degrees_east  circular',
```

```
 'lat : 88.57217 to -88.57217 degrees_north',
 'Vertical coordinates :',
 '1 : surface                   : levels=1',
 'Time coordinate :',
 'time : 40 steps',
 'RefTime =  2001-01-01 00:00:00  Units = hours  Calendar = standard',
 'YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD
hh:mm:ss',
 '2001-01-01 00:00:00  2001-01-01 06:00:00  2001-01-01 12:00:00  2001-01-01
18:00:00',
 '2001-01-02 00:00:00  2001-01-02 06:00:00  2001-01-02 12:00:00  2001-01-02
18:00:00',
 '2001-01-03 00:00:00  2001-01-03 06:00:00  2001-01-03 12:00:00  2001-01-03
18:00:00',
 '2001-01-04 00:00:00  2001-01-04 06:00:00  2001-01-04 12:00:00  2001-01-04
18:00:00',
 '2001-01-05 00:00:00  2001-01-05 06:00:00  2001-01-05 12:00:00  2001-01-05
18:00:00',
 '2001-01-06 00:00:00  2001-01-06 06:00:00  2001-01-06 12:00:00  2001-01-06
18:00:00',
 '2001-01-07 00:00:00  2001-01-07 06:00:00  2001-01-07 12:00:00  2001-01-07
18:00:00',
 '2001-01-08 00:00:00  2001-01-08 06:00:00  2001-01-08 12:00:00  2001-01-08
18:00:00',
 '2001-01-09 00:00:00  2001-01-09 06:00:00  2001-01-09 12:00:00  2001-01-09
18:00:00',
 '2001-01-10 00:00:00  2001-01-10 06:00:00  2001-01-10 12:00:00  2001-01-10
18:00:00']
```

1.15 Select timesteps

Select timestep 1 and 10:

[22]:
```
cdo.seltimestep('1,10', input=infile, output=outfile)
cdo.sinfon(input=outfile)
```

[22]:
```
['File format : NetCDF',
 '-1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter
name',
 '1 : unknown  unknown  v instant       1   1     18432   1  F32  : tsurf',
 '2 : unknown  unknown  v instant       1   1     18432   1  F32  : precip',
 '3 : unknown  unknown  v instant       1   1     18432   1  F32  : u10',
 '4 : unknown  unknown  v instant       1   1     18432   1  F32  : v10',
 '5 : unknown  unknown  v instant       1   1     18432   1  F32  : qvi',
 '6 : unknown  unknown  v instant       1   1     18432   1  F32  : slp',
 'Grid coordinates :',
```

```
 '1 : gaussian                 : points=18432 (192x96)',
 'lon : -180 to 178.125 by 1.875 degrees_east  circular',
 'lat : 88.57217 to -88.57217 degrees_north',
 'Vertical coordinates :',
 '1 : surface                  : levels=1',
 'Time coordinate :',
 'time : 2 steps',
 'RefTime =  2001-01-01 00:00:00  Units = hours  Calendar = standard',
 'YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD
hh:mm:ss',
 '2001-01-01 00:00:00  2001-01-03 06:00:00']
```

Select timestep 1 to 10:

[23]:
```
cdo.seltimestep('1/10', input=infile, output=outfile)
cdo.sinfon(input=outfile)
```

[23]:
```
['File format : NetCDF',
 '-1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter
name',
 '1 : unknown  unknown  v instant       1   1     18432   1  F32  : tsurf',
 '2 : unknown  unknown  v instant       1   1     18432   1  F32  : precip',
 '3 : unknown  unknown  v instant       1   1     18432   1  F32  : u10',
 '4 : unknown  unknown  v instant       1   1     18432   1  F32  : v10',
 '5 : unknown  unknown  v instant       1   1     18432   1  F32  : qvi',
 '6 : unknown  unknown  v instant       1   1     18432   1  F32  : slp',
 'Grid coordinates :',
 '1 : gaussian                 : points=18432 (192x96)',
 'lon : -180 to 178.125 by 1.875 degrees_east  circular',
 'lat : 88.57217 to -88.57217 degrees_north',
 'Vertical coordinates :',
 '1 : surface                  : levels=1',
 'Time coordinate :',
 'time : 10 steps',
 'RefTime =  2001-01-01 00:00:00  Units = hours  Calendar = standard',
 'YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD
hh:mm:ss',
 '2001-01-01 00:00:00  2001-01-01 06:00:00  2001-01-01 12:00:00  2001-01-01
18:00:00',
 '2001-01-02 00:00:00  2001-01-02 06:00:00  2001-01-02 12:00:00  2001-01-02
18:00:00',
 '2001-01-03 00:00:00  2001-01-03 06:00:00']
```

1.16 Operator chaining

Operators with a fixed number of input files (streams) and only one output file can be combined. The input parameters must begin with an '-' and they will be executed from right to left.

Select the variables u10 and v10 and then select the first 10 timesteps:

[24]:
```
cdo.seltimestep('1/10', input='-selvar,u10,v10 '+infile, output=outfile)
cdo.sinfon(input=outfile)
```

[24]:
```
['File format : NetCDF',
 '-1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter
name',
 '1 : unknown  unknown  v instant       1   1     18432   1  F32  : u10',
 '2 : unknown  unknown  v instant       1   1     18432   1  F32  : v10',
 'Grid coordinates :',
 '1 : gaussian                  : points=18432 (192x96)',
 'lon : -180 to 178.125 by 1.875 degrees_east  circular',
 'lat : 88.57217 to -88.57217 degrees_north',
 'Vertical coordinates :',
 '1 : surface                   : levels=1',
 'Time coordinate :',
 'time : 10 steps',
 'RefTime =  2001-01-01 00:00:00  Units = hours  Calendar = standard',
 'YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD
hh:mm:ss',
 '2001-01-01 00:00:00  2001-01-01 06:00:00  2001-01-01 12:00:00  2001-01-01
18:00:00',
 '2001-01-02 00:00:00  2001-01-02 06:00:00  2001-01-02 12:00:00  2001-01-02
18:00:00',
 '2001-01-03 00:00:00  2001-01-03 06:00:00']
```

Use operators and options at once to do the above sections and change the output precision:

[25]:
```
cdo.seltimestep('1/10', input='-selvar,u10,v10 '+infile, options='-b F64',␣
↪output=outfile)
cdo.sinfon(input=outfile)
```

[25]:
```
['File format : NetCDF',
 '-1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter
name',
 '1 : unknown  unknown  v instant       1   1     18432   1  F64  : u10',
 '2 : unknown  unknown  v instant       1   1     18432   1  F64  : v10',
 'Grid coordinates :',
 '1 : gaussian                  : points=18432 (192x96)',
 'lon : -180 to 178.125 by 1.875 degrees_east  circular',
 'lat : 88.57217 to -88.57217 degrees_north',
 'Vertical coordinates :',
 '1 : surface                   : levels=1',
 'Time coordinate :',
 'time : 10 steps',
 'RefTime =  2001-01-01 00:00:00  Units = hours  Calendar = standard',
 'YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD
```

```
hh:mm:ss',
 '2001-01-01 00:00:00  2001-01-01 06:00:00  2001-01-01 12:00:00  2001-01-01
18:00:00',
 '2001-01-02 00:00:00  2001-01-02 06:00:00  2001-01-02 12:00:00  2001-01-02
18:00:00',
 '2001-01-03 00:00:00  2001-01-03 06:00:00']
```

1.17 Select a sub-region

[26]:
```
cdo.sellonlatbox('20,30,70,80', input='-seltimestep,1 '+infile, output=outfile)
cdo.sinfon(input=outfile)
```

[26]:
```
['File format : NetCDF',
 '-1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter
name',
 '1 : unknown  unknown  v instant       1   1        30   1  F32  : tsurf',
 '2 : unknown  unknown  v instant       1   1        30   1  F32  : precip',
 '3 : unknown  unknown  v instant       1   1        30   1  F32  : u10',
 '4 : unknown  unknown  v instant       1   1        30   1  F32  : v10',
 '5 : unknown  unknown  v instant       1   1        30   1  F32  : qvi',
 '6 : unknown  unknown  v instant       1   1        30   1  F32  : slp',
 'Grid coordinates :',
 '1 : gaussian                 : points=30 (6x5)',
 'lon : 20.625 to 30 by 1.875 degrees_east',
 'lat : 79.27056 to 71.81113 degrees_north',
 'Vertical coordinates :',
 '1 : surface                  : levels=1',
 'Time coordinate :',
 'time : 1 step',
 'RefTime =  2001-01-01 00:00:00  Units = hours  Calendar = standard',
 'YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD
hh:mm:ss',
 '2001-01-01 00:00:00']
```

1.18 Compute the field mean

Compute the mean of the horizontal field for each timestep (-> time series).

[27]:
```
cdo.fldmean(input="-selname,tsurf "+infile, output='outfile.nc')
cdo.sinfon(input=outfile)
```

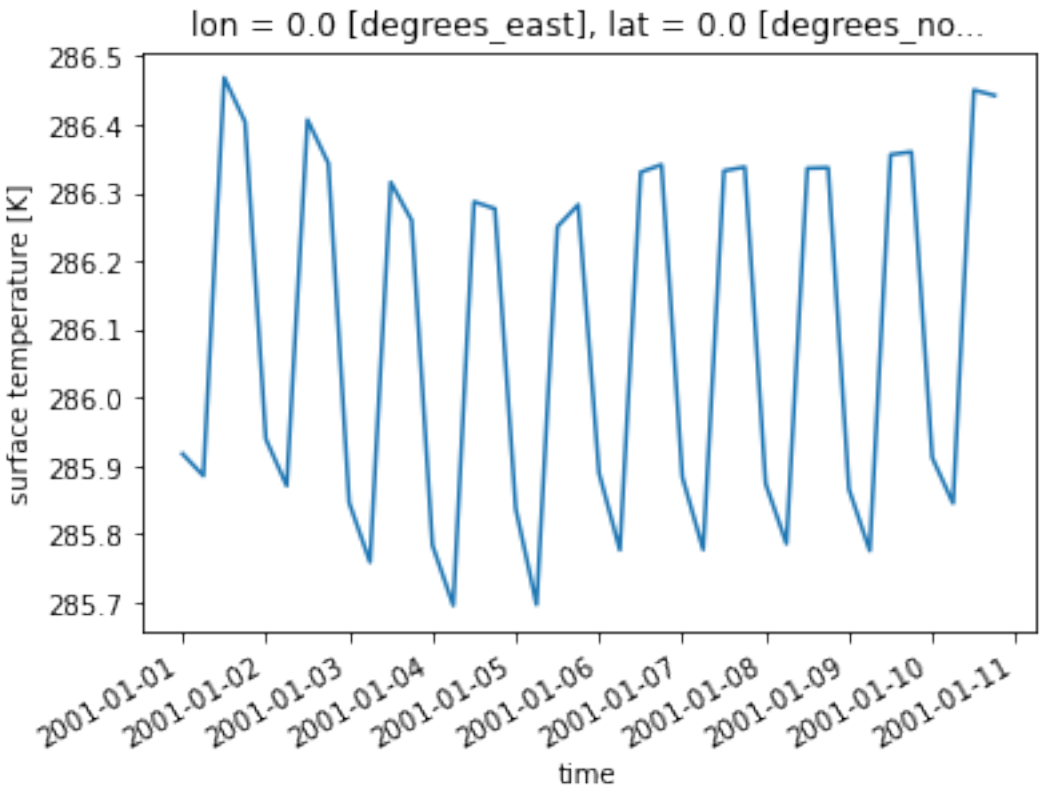
[27]:
```
['File format : NetCDF',
 '-1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter
name',
 '1 : unknown  unknown  v instant       1   1         1   1  F32  : tsurf',
```

```
 'Grid coordinates :',
 '1 : lonlat                    : points=1 (1x1)',
 'lon : 0 degrees_east',
 'lat : 0 degrees_north',
 'Vertical coordinates :',
 '1 : surface                   : levels=1',
 'Time coordinate :',
 'time : 40 steps',
 'RefTime =  2001-01-01 00:00:00  Units = hours  Calendar = standard',
 'YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD hh:mm:ss  YYYY-MM-DD
hh:mm:ss',
 '2001-01-01 00:00:00  2001-01-01 06:00:00  2001-01-01 12:00:00  2001-01-01
18:00:00',
 '2001-01-02 00:00:00  2001-01-02 06:00:00  2001-01-02 12:00:00  2001-01-02
18:00:00',
 '2001-01-03 00:00:00  2001-01-03 06:00:00  2001-01-03 12:00:00  2001-01-03
18:00:00',
 '2001-01-04 00:00:00  2001-01-04 06:00:00  2001-01-04 12:00:00  2001-01-04
18:00:00',
 '2001-01-05 00:00:00  2001-01-05 06:00:00  2001-01-05 12:00:00  2001-01-05
18:00:00',
 '2001-01-06 00:00:00  2001-01-06 06:00:00  2001-01-06 12:00:00  2001-01-06
18:00:00',
 '2001-01-07 00:00:00  2001-01-07 06:00:00  2001-01-07 12:00:00  2001-01-07
18:00:00',
 '2001-01-08 00:00:00  2001-01-08 06:00:00  2001-01-08 12:00:00  2001-01-08
18:00:00',
 '2001-01-09 00:00:00  2001-01-09 06:00:00  2001-01-09 12:00:00  2001-01-09
18:00:00',
 '2001-01-10 00:00:00  2001-01-10 06:00:00  2001-01-10 12:00:00  2001-01-10
18:00:00']
```

Plot the fieldmean data:

[28]:
```
cdo.fldmean(input="-selname,tsurf "+infile, returnXArray='tsurf').plot()
```

[28]:
```
[<matplotlib.lines.Line2D at 0x182707e50>]
```



1.19 Assign data to variable - examples

1. Assign the file variable *precip* to a python variable:

[29]:
```
precipitation = cdo.selvar('precip', input=infile, returnXArray='precip')
print(precipitation)
```

```
<xarray.DataArray 'precip' (time: 40, lat: 96, lon: 192)>
[737280 values with dtype=float32]
Coordinates:
  * time     (time) datetime64[ns] 2001-01-01 … 2001-01-10T18:00:00
  * lon      (lon) float64 -180.0 -178.1 -176.2 -174.4 … 174.4 176.2 178.1
  * lat      (lat) float64 88.57 86.72 84.86 83.0 … -83.0 -84.86 -86.72 -88.57
Attributes:
    long_name:      total precipitation
    units:          kg/m^2s
    code:           4
    table:          128
    CDI_grid_type:  gaussian
```

2. Compute the fieldmean of the variable *tsurf*:

[30]:
```
tsurf_fldmean = cdo.fldmean(input=infile, returnXArray='tsurf')
print(tsurf_fldmean.values[0:10,0,0])
```

```
[285.91788 285.8857  286.46826 286.4037  285.9395  285.8711  286.40665
 286.34302 285.84647 285.7593 ]
```

3. Convert the units from `kg/m^2s` to `mm/day` by multiplying with 86400 (60 seconds * 60 minutes * 24 hours).

Note: Each time a calculation is performed with XArrays, the array attributes are cleared. It's up to you to restore them correctly afterwards.

[31]:
```
tsurf_fldmean = tsurf_fldmean * 86400
tsurf_fldmean.attrs['units'] = 'mm/day'
print(tsurf_fldmean)
```

```
<xarray.DataArray 'tsurf' (time: 40, lat: 1, lon: 1)>
array([[[24703304.58984375]],

       [[24700525.48828125]],

       [[24750857.8125    ]],

       [[24745278.515625  ]],

       [[24705174.0234375 ]],

       [[24699262.5       ]],

       [[24745534.27734375]],

       [[24740036.71875   ]],

       [[24697134.66796875]],

       [[24689604.19921875]],

...

       [[24739395.99609375]],

       [[24739477.734375  ]],

       [[24698835.3515625 ]],

       [[24691057.03125   ]],

       [[24741130.95703125]],
```

```
       [[24741471.09375   ]],

       [[24702816.796875  ]],

       [[24697055.56640625]],

       [[24749288.96484375]],

       [[24748603.41796875]]])
Coordinates:
  * time     (time) datetime64[ns] 2001-01-01 … 2001-01-10T18:00:00
  * lon      (lon) float64 0.0
  * lat      (lat) float64 0.0
Attributes:
    units:    mm/day
```

1.20 Remapping

Interpolate the data of input file to a new grid using the bilinear interpolation method.

For the ease of use, we select the variable tsurf and only the first timestep to interpolate the data to a longitude 1 deg x latitude 1 deg (r360x180) grid.

[32]:
```
cdo.remapbil('r360x180', input='-seltimestep,1 -selvar,tsurf '+infile,␣
 ↪output=outfile)
```

[32]:
```
'./outfile.nc'
```

To demonstrate the functionality we increase the resolution of the input data to a 0.5 deg grid and plot the original and the interpolated data.

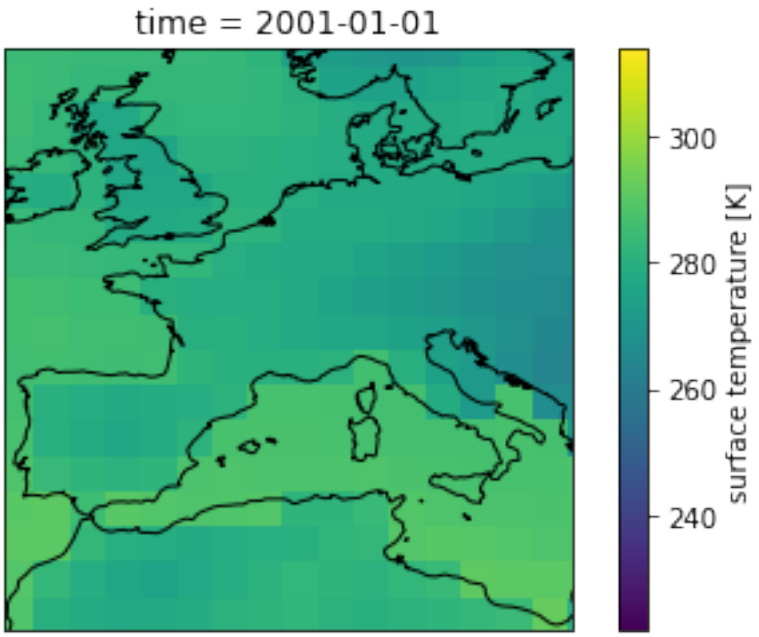
Original input data:

[33]:
```
import cartopy.crs as ccrs
import xarray as xr
tsurf_orig = xr.open_dataset(infile).tsurf[0,:,:]

data = tsurf_orig.plot(subplot_kws=dict(projection=ccrs.PlateCarree(),␣
 ↪facecolor='gray'), transform=ccrs.PlateCarree(),)

data.axes.set_extent([-10.,20.,30.,60.])
data.axes.coastlines()
```

[33]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x182782230>
```

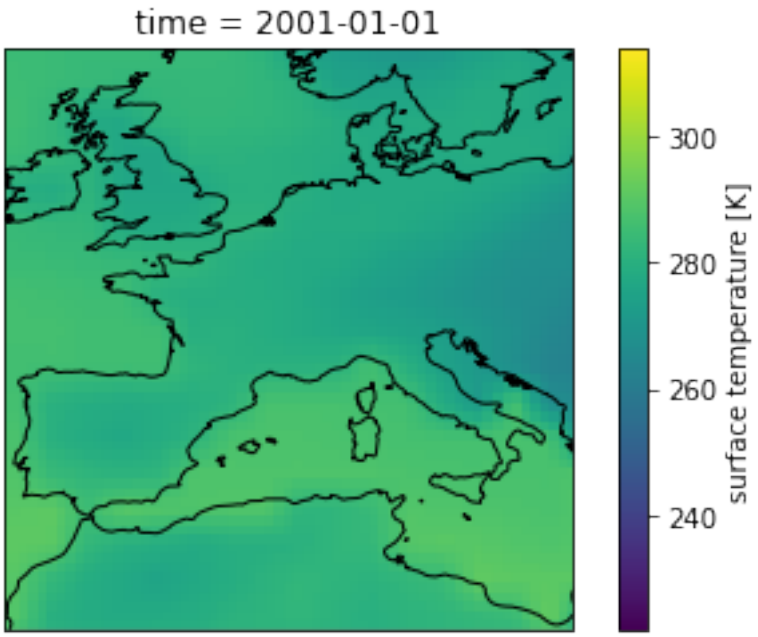


[34]:
```
tsurf = cdo.remapbil('r720x360', input='-seltimestep,1 '+infile,␣
 ↪returnXArray='tsurf')

data = tsurf.plot(subplot_kws=dict(projection=ccrs.PlateCarree(),␣
 ↪facecolor='gray'), transform=ccrs.PlateCarree(),)

data.axes.set_extent([-10.,20.,30.,60.])
data.axes.coastlines()
```

[34]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x1846f1930>
```



1.21 Remap data to a grid of another file

[35]:
```
cdo.topo(options='-f nc', output='topo.nc')

tsurf_2 = cdo.remapbil('topo.nc', input="-seltimestep,1 "+infile,␣
 ↪returnXArray='tsurf')
```

1.22 Create a land-sea mask

CDO provides a global 0.5 degree topography dataset that can be used to generate a land sea mask. First, we interpolate the topography data to the same grid as the data file. With the operator `gtc` we set all values greater than 0.5 m to 1 and all other values to 0.

[36]:
```
lsm = cdo.gtc(0.5, input='-remapbil,'+infile+' -topo', returnXArray='topo')
print(lsm)
```

```
<xarray.DataArray 'topo' (lat: 96, lon: 192)>
array([[0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [1., 1., 1., …, 1., 1., 1.],
       [1., 1., 1., …, 1., 1., 1.],
       [1., 1., 1., …, 1., 1., 1.]], dtype=float32)
```

```
Coordinates:
  * lon      (lon) float64 -180.0 -178.1 -176.2 -174.4 … 174.4 176.2 178.1
  * lat      (lat) float64 88.57 86.72 84.86 83.0 … -83.0 -84.86 -86.72 -88.57
Attributes:
    units:          m
    CDI_grid_type:  gaussian
```

[37]:
```
p = lsm.plot(subplot_kws=dict(projection=ccrs.PlateCarree(), facecolor='gray'),␣
 ↪transform=ccrs.PlateCarree(),)
p.axes.set_extent([-20.,20.,30.,60.])
p.axes.coastlines()
```

[37]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x1877f0760>
```

1.23 Masking data

Before we can use a mask on a data variable we need to create a mask file using the same grid as the data variable.

[38]:
```
cdo.setname('lsm', input='-gtc,0.5 -remapbil,'+infile+' -topo', options='-f␣
 ↪nc', output='lsm.nc')
```

[38]:
```
'lsm.nc'
```

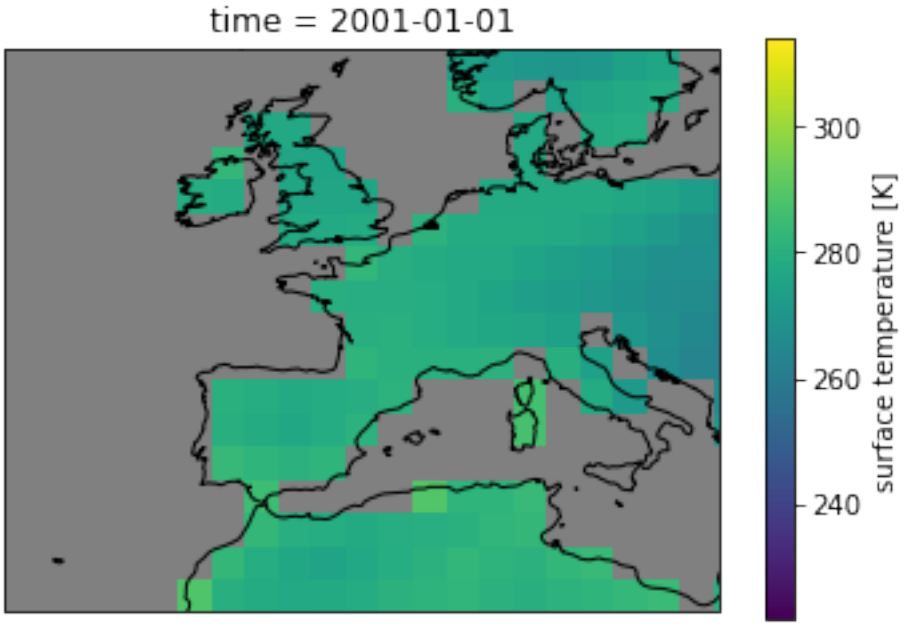
Now, we want to get only tsurf values over land for timestep 1.

[39]:
```
masked = cdo.setctomiss(0, input='-mul lsm.nc -seltimestep,1 '+infile,␣
↪returnXArray='tsurf')
```

Let's see how it looks like.

[40]:
```
p = masked.plot(subplot_kws=dict(projection=ccrs.PlateCarree(),␣
↪facecolor='gray'), transform=ccrs.PlateCarree(),)
p.axes.set_extent([-20.,20.,30.,60.])
p.axes.coastlines()
```

[40]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x1878b4610>
```



Use a variables instead of output files.

[41]:
```
lsm = cdo.setname('lsm', input='-gtc,0.5 -remapbil,'+infile+' -topo',␣
↪options='-f nc', returnXArray='lsm')
```

[42]:
```
tsurf = cdo.seltimestep(1, input='-selvar,tsurf '+infile, returnXArray='tsurf')
```

[43]:
```
masked = tsurf * lsm
print(masked)
```

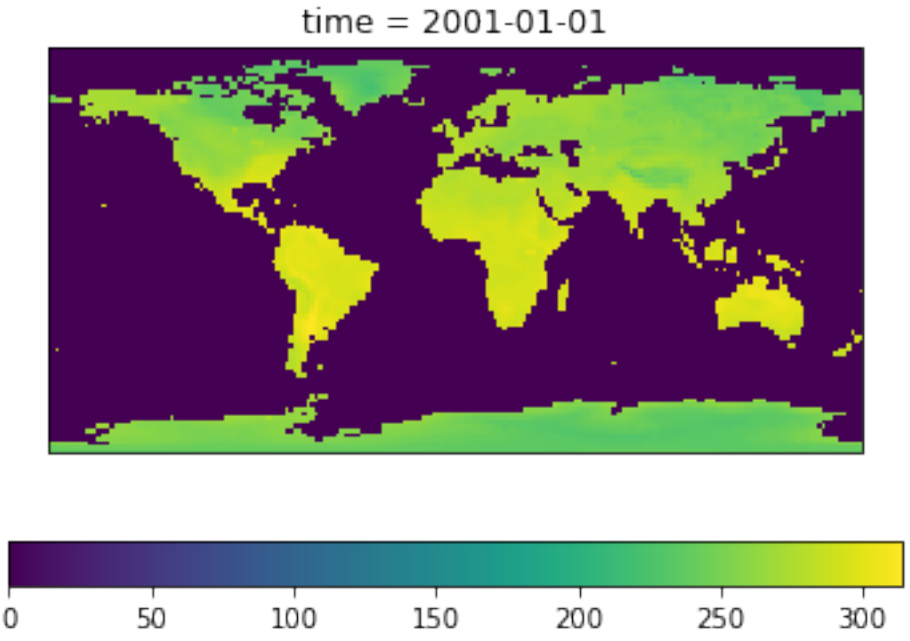
```
<xarray.DataArray (time: 1, lat: 96, lon: 192)>
array([[[  0.      ,   0.      ,   0.      , …,   0.      ,   0.      ,
           0.      ],
        [  0.      ,   0.      ,   0.      , …,   0.      ,   0.      ,
           0.      ],
```

```
       [  0.      ,   0.      ,   0.      , …,   0.      ,   0.      ,
          0.      ],
       …,
       [256.27307, 256.78674, 257.43127, …, 254.5895 , 255.06606,
        255.69496],
       [242.54457, 242.53676, 242.91371, …, 241.52113, 241.91566,
        242.33168],
       [236.11879, 235.98012, 235.96059, …, 236.11488, 236.09145,
        236.07191]]], dtype=float32)
Coordinates:
  * time     (time) datetime64[ns] 2001-01-01
  * lon      (lon) float64 -180.0 -178.1 -176.2 -174.4 … 174.4 176.2 178.1
  * lat      (lat) float64 88.57 86.72 84.86 83.0 … -83.0 -84.86 -86.72 -88.57
```

Plot the masked data.

[44]:
```
masked.plot(subplot_kws=dict(projection=ccrs.PlateCarree(), facecolor='gray'),␣
 ↪transform=ccrs.PlateCarree(),
           cbar_kwargs={'orientation': 'horizontal'})
p.axes.set_extent([-20.,20.,30.,60.])
p.axes.coastlines()
```

[44]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x184753040>
```



And now we want to mask the land part of the data. The easiest way is to set all zeros to one and all ones to NaN. Matplotlib will automatically use the color *grey* for the missing values.

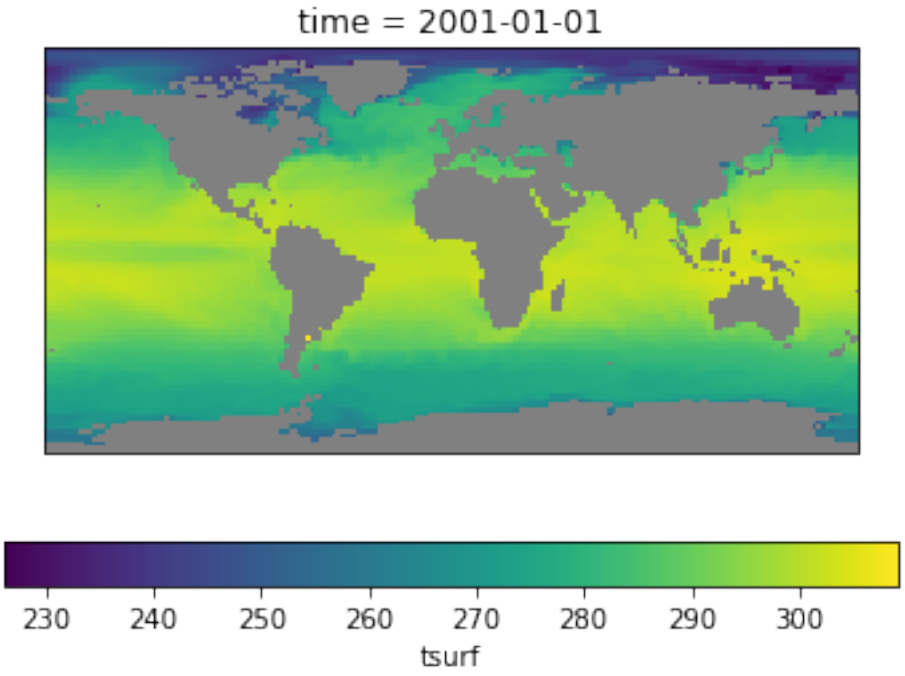
[45]:
```
import numpy as np
import matplotlib as mpl
import copy

lsm = np.where(lsm==0, 1, np.nan)

masked = tsurf * lsm

masked.plot(subplot_kws=dict(projection=ccrs.PlateCarree(), facecolor='gray'),␣
↪transform=ccrs.PlateCarree(),
           cbar_kwargs={'orientation': 'horizontal'})
p.axes.set_extent([-20.,20.,30.,60.])
p.axes.coastlines()
```

[45]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x1847521d0>
```



1.24 Delete temporary files

[46]:
```
cdo.cleanTempDir()
```