Interactive visualization of climate model data via Python or GUI with psyplot

DACH 2022

Philipp S. Sommer

Helmholtz-Zentrum Hereon, Helmholtz Coastal Data Center

March 23rd, 2022

Help

Technical Note

This presentation is a jupyter notebook presented with rise for interactive execution of the cells. You can run it interactively on mybinder in your browser:

launch binder

The link to the repo on Github: https://github.com/Chilipp/psyplot-DACH2022-presentation).

Back to first slide

Technical Note

This presentation is a jupyter notebook presented with rise for interactive execution of the cells. You can run it interactively on mybinder in your browser:

launch binder

The link to the repo on Github: https://github.com/Chilipp/psyplot-DACH2022-presentation).

Back to first slide

So let's import some libraries for the execution

In [1]:
```
%matplotlib widget

import psyplot.project as psy

import numpy as np
import xarray as xr
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
from IPython.display import display, Video

from ipympl.backend_nbagg import Canvas
Canvas.header_visible.default_value = False

import pyvista
pyvista.rcParams['use_ipyvtk'] = True

import warnings
warnings.filterwarnings("ignore", r"\s*The on_mappable_chang
warnings.filterwarnings("ignore", r"\s*The input coordinates
warnings.filterwarnings("ignore", r"\s*shading=")
warnings.filterwarnings("ignore", r"\s*\[Warning by")
```

Outline

Main features of psyplot

How to use and extend the framework

Some more features of psyplot

Outline

Main features of psyplot

How to use and extend the framework

Some more features of psyplot

Note:

I am not a visualization expert

The aim of this talk is not to show wonderful plots, but rather how to generate them.

You can always make them publication-ready using the rich features of matplotlib.

Main features of psyplot

Using psyplot from Python

In [2]:
```
ds = psy.open_dataset("data/icon_grid_demo.nc")
ds
```

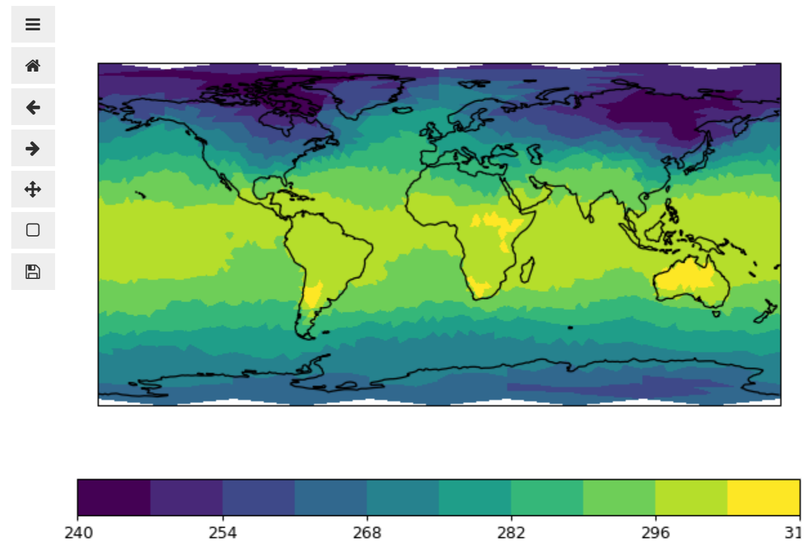
Out[2]:
```
<xarray.Dataset>
Dimensions:    (time: 5, ncells: 5120, vertices: 3, edge:
480, no: 4, lev: 4)
Coordinates:
  * time       (time) datetime64[ns] 1979-01-31T18:00:00
... 1979-05-31T18:00:00
    clon       (ncells) float64 ...
    clon_bnds  (ncells, vertices) float64 ...
    clat       (ncells) float64 ...
    clat_bnds  (ncells, vertices) float64 ...
    elon       (edge) float32 ...
    elon_bnds  (edge, no) float32 ...
    elat       (edge) float32 ...
    elat_bnds  (edge, no) float32 ...
  * lev        (lev) float64 1e+05 8.5e+04 5e+04 2e+04
Dimensions without coordinates: ncells, vertices, edge, no
Data variables:
    t2m        (time, lev, ncells) float32 ...
    u          (time, lev, ncells) float32 ...
    v          (time, lev, ncells) float32 ...
    t2m_edge   (time, lev, edge) float32 ...
Attributes:
    CDI:                  Climate Data Interface version
1.9.1 (http://mpimet...
    Conventions:          CF-1.4
    history:              Thu Aug 30 21:54:23 2018: cdo de
lname,t2m_2d,u_2d,v...
    number_of_grid_used:  42
    uuidOfHGrid:          bf575ad8-daa6-11e7-a4a9-93d511f8
21b4
    title:                Temperature and Wind demo-File f
or python nc2map mo...
    CDO:                  Climate Data Operators version
1.9.1 (http://mpimet...
```

Using psyplot from Python

In [2]:
```
ds = psy.open_dataset("data/icon_grid_demo.nc")
ds
```

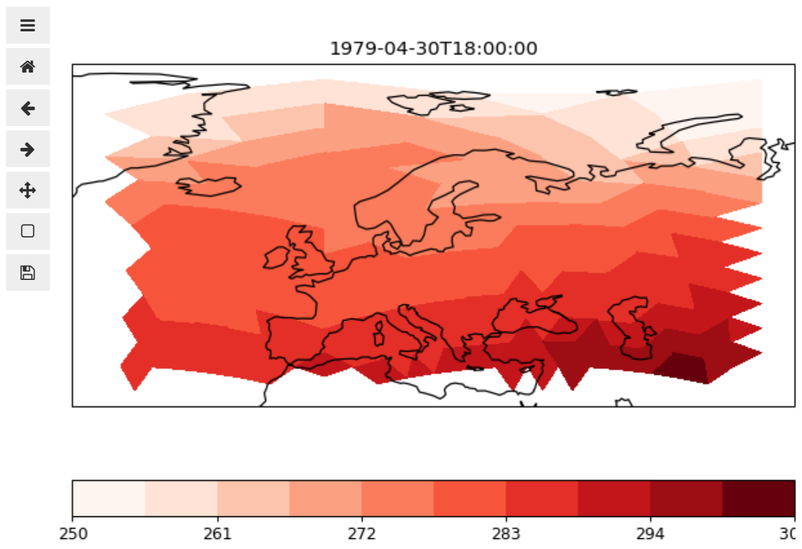
Out[2]:
```
<xarray.Dataset>
Dimensions:    (time: 5, ncells: 5120, vertices: 3, edge:
480, no: 4, lev: 4)
Coordinates:
  * time       (time) datetime64[ns] 1979-01-31T18:00:00
... 1979-05-31T18:00:00
    clon       (ncells) float64 ...
    clon_bnds  (ncells, vertices) float64 ...
    clat       (ncells) float64 ...
    clat_bnds  (ncells, vertices) float64 ...
    elon       (edge) float32 ...
    elon_bnds  (edge, no) float32 ...
    elat       (edge) float32 ...
    elat_bnds  (edge, no) float32 ...
  * lev        (lev) float64 1e+05 8.5e+04 5e+04 2e+04
Dimensions without coordinates: ncells, vertices, edge, no
Data variables:
    t2m        (time, lev, ncells) float32 ...
    u          (time, lev, ncells) float32 ...
    v          (time, lev, ncells) float32 ...
    t2m_edge   (time, lev, edge) float32 ...
Attributes:
    CDI:                  Climate Data Interface version
1.9.1 (http://mpimet...
    Conventions:          CF-1.4
    history:              Thu Aug 30 21:54:23 2018: cdo de
lname,t2m_2d,u_2d,v...
    number_of_grid_used:  42
    uuidOfHGrid:          bf575ad8-daa6-11e7-a4a9-93d511f8
21b4
    title:                Temperature and Wind demo-File f
or python nc2map mo...
    CDO:                  Climate Data Operators version
1.9.1 (http://mpimet...
```

In [3]:
```
sp = ds.psy.plot.mapplot(
    name="t2m",
)
```



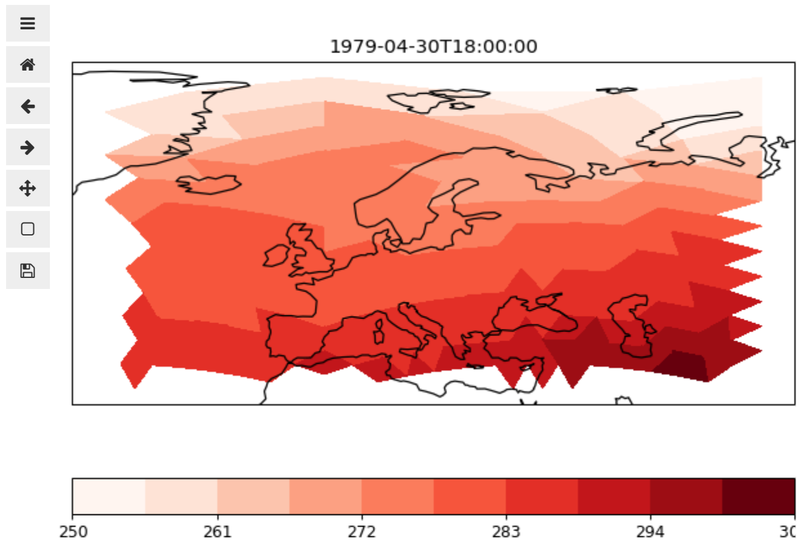
Working interactively from the command line

In [4]:
```
sp = ds.psy.plot.mapplot(
    name="t2m", cmap="Blues",
)
```



Working interactively from the command line

```
In [4]: sp = ds.psy.plot.mapplot(
            name="t2m", cmap="Blues",
        )
```




```
In [5]: sp.update(cmap="Reds")
```

```
In [6]: sp.update(title="%(time)s")
```

```
In [7]: sp.update(time=3)
```

```
In [8]: sp.update(lonlatbox="Europe")
```

```
In [9]: psy.close("all")
```

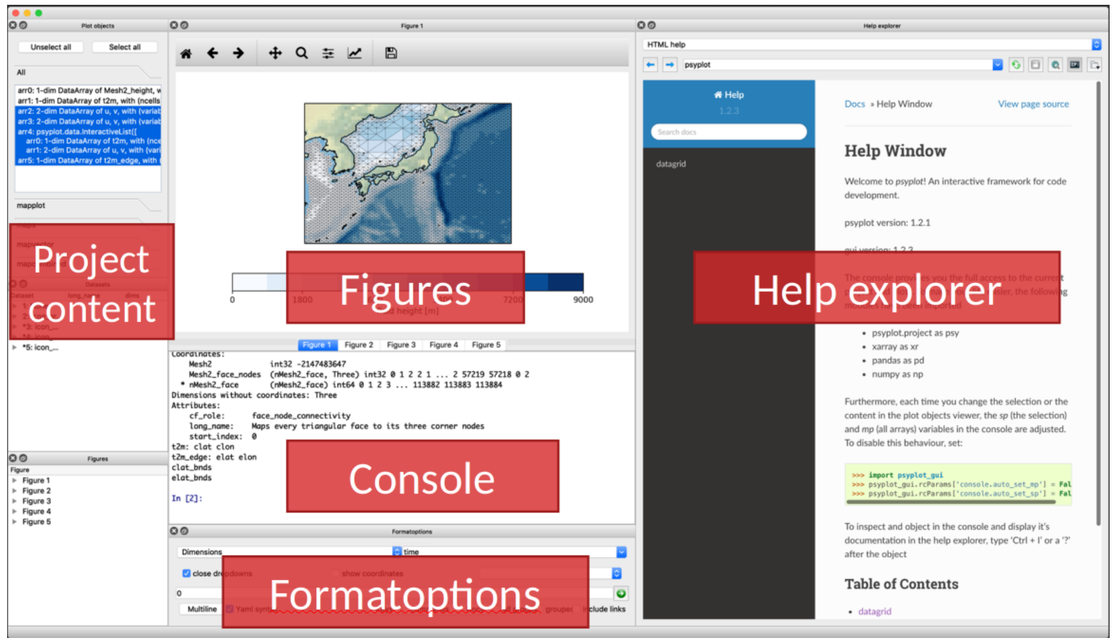
Using the GUI

psyplot comes with a flexible graphical user interface (GUI).

- On mybinder: click here.
- On mistral:
  - either via X11

    ```
    ssh -X mistral
    module load python3
    psyplot
    ```

  - or via remote desktop
- On your on own working station: Install it via `conda install -c conda-forge psy-view`

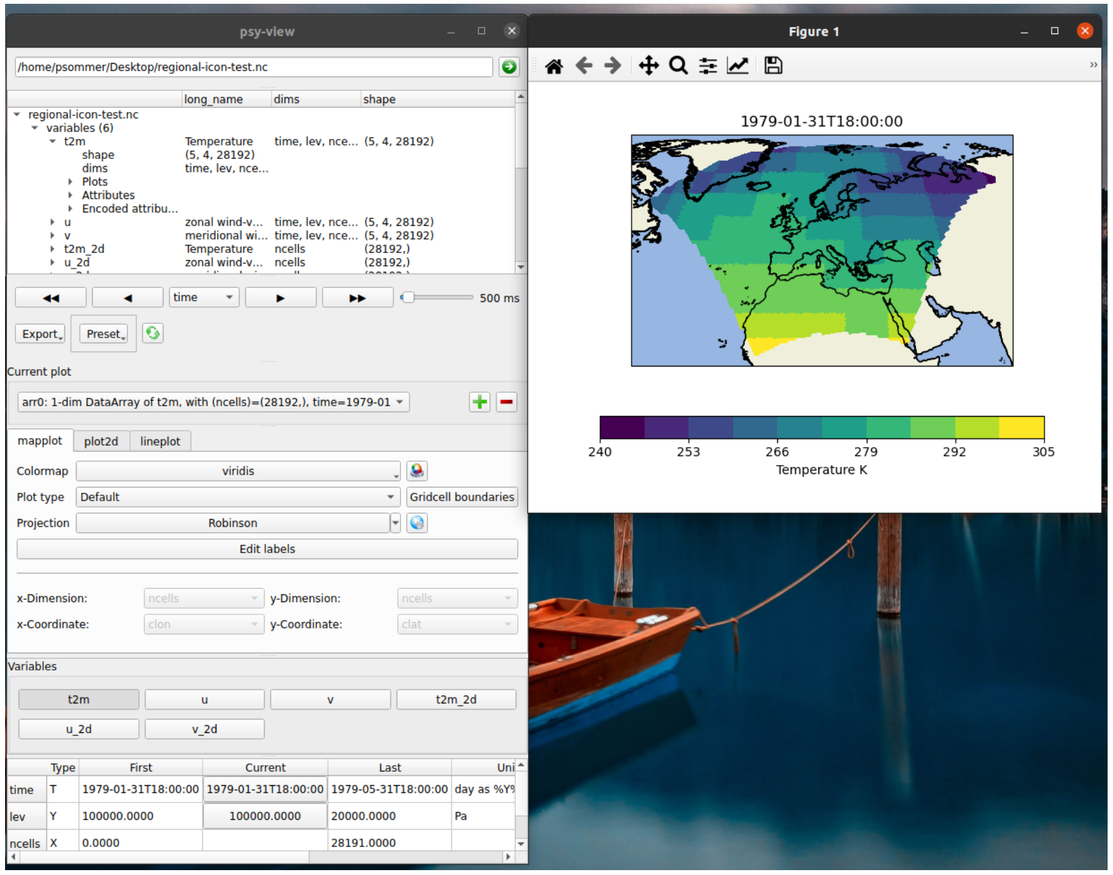


psy-view

An ncview-like interface, but with psyplot

- quick access to netcdf-variables via buttons
- switch between projections
- modify basemap
- change labels, colormaps, etc.
- display time-series when clicking on the map
- load presets for your plots
- animate through time, z, etc.

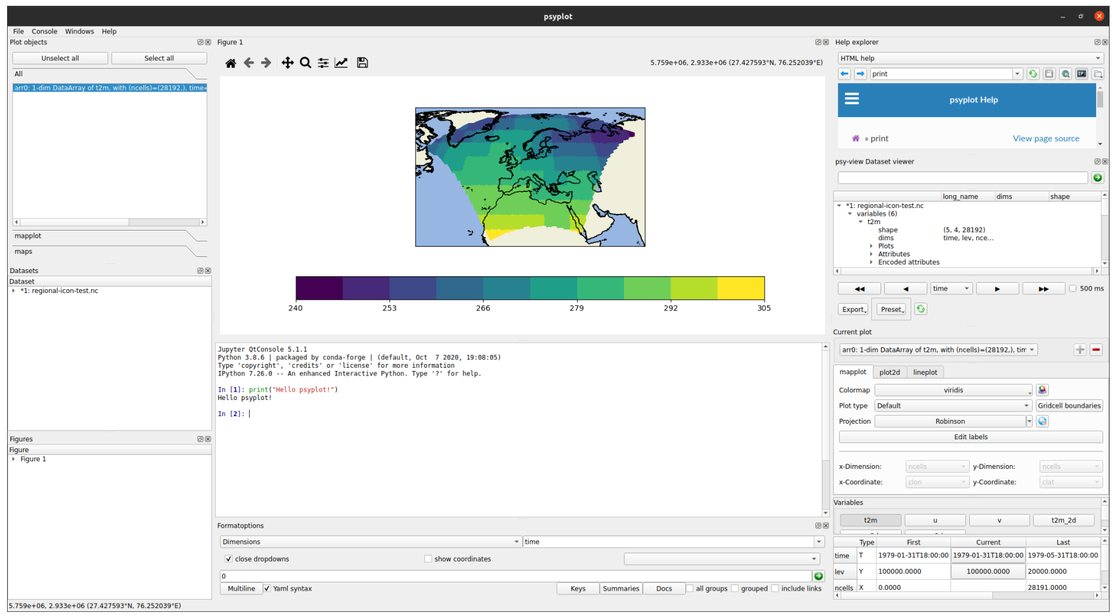
https://psyplot.github.io/psy-view



The psyplot GUI

Flexible GUI for coding and clicking

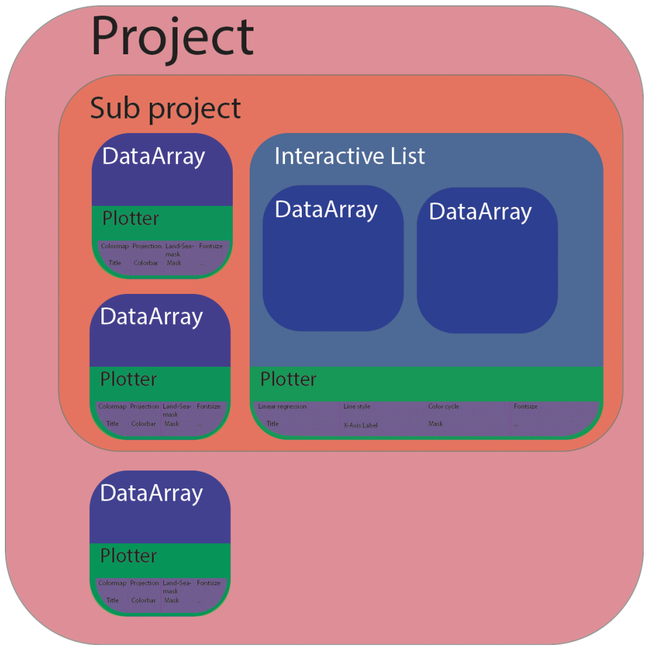
- integrated IPython console for interactive use of the command line
- connected help window that renders help and python object documentation
- integrated psy-view window
- shortcut widgets for individual formatoptions



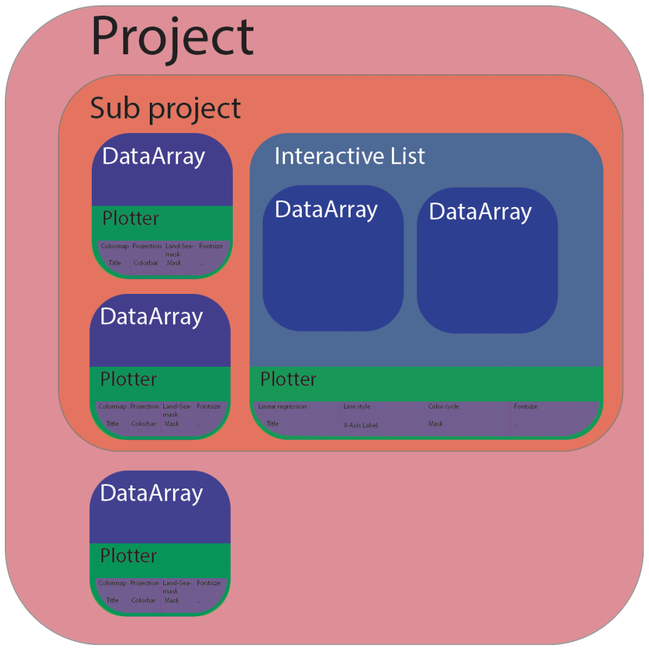
https://psyplot.github.io/psyplot-gui

An introduction into the psyplot framework

The psyplot data model

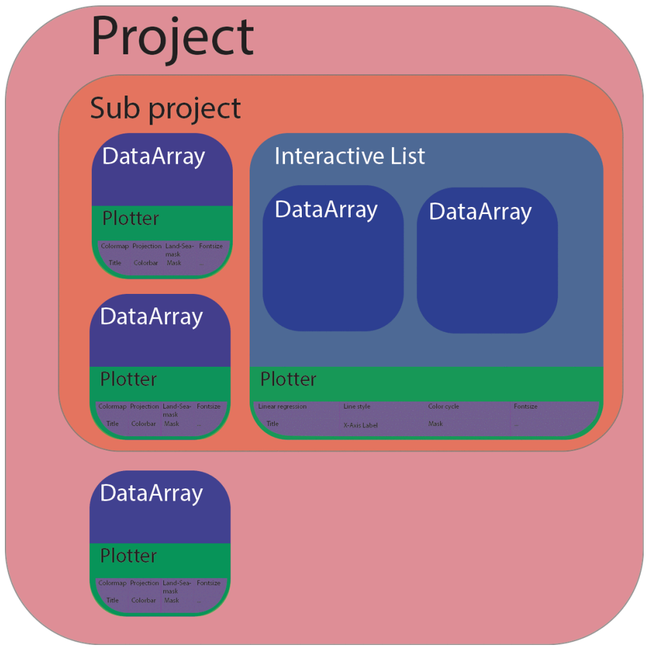


The psyplot data model



- `Formatoption`: The smallest possible unit. Each formatoption controls one aspect of the plot (cmap, lonlatbox, etc.)
- `Plotter`: A set of formatoptions that visualizes data

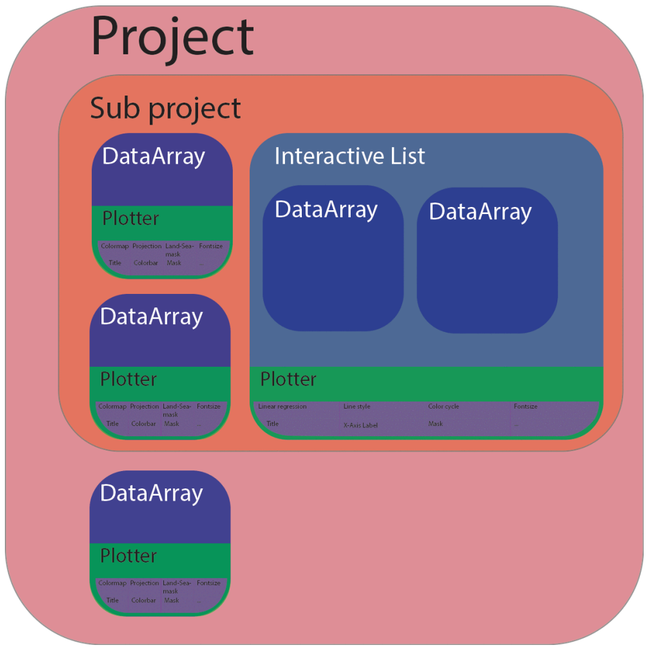
The psyplot data model



- `Formatoption`: The smallest possible unit. Each formatoption controls one aspect of the plot (cmap, lonlatbox, etc.)
- `Plotter`: A set of formatoptions that visualizes data

- `DataArray`: Standard `xarray`
- `InteractiveList`: A collection of `DataArray`s that are visualized by one plotter (e.g. a collection of lines)

The psyplot data model



- `Formatoption`: The smallest possible unit. Each formatoption controls one aspect of the plot (cmap, lonlatbox, etc.)
- `Plotter`: A set of formatoptions that visualizes data

- `DataArray`: Standard `xarray`
- `InteractiveList`: A collection of `DataArray`s that are visualized by one plotter (e.g. a collection of lines)

- `Project`: A set of data objects, each visualized by one plotter
- `Sub project`: A subset of a larger `Project`

Low-level interface

Low-level interface

import the necessary objects from the framework

In [10]:
```
from psyplot.plotter import Formatoption, Plotter
```

Low-level interface

import the necessary objects from the framework

In [10]:
```
from psyplot.plotter import Formatoption, Plotter
```

define a formatoption

In [11]:
```
class MyFormatoption(Formatoption):

    #: the default value for the formatoption
    default = 'my text'

    def initialize_plot(self, value):
        # method initialize the plot in the very beginning
        self.text = self.ax.text(0.5, 0.5, value, fontsize="xx-large")

    def update(self, value):
        # method to update the plot
        self.text.set_text(value)
```

Low-level interface

import the necessary objects from the framework

In [10]:
```
from psyplot.plotter import Formatoption, Plotter
```

define a formatoption

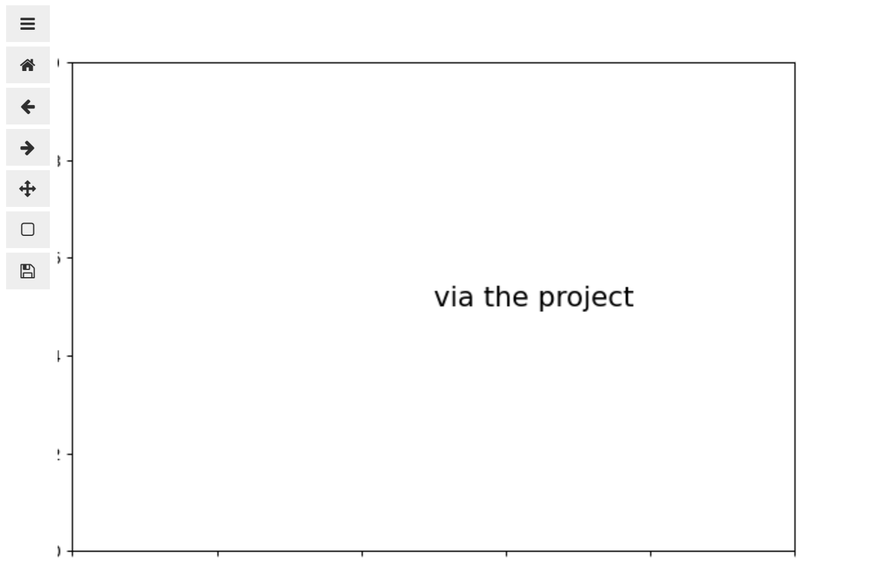
In [11]:
```
class MyFormatoption(Formatoption):

    #: the default value for the formatoption
    default = 'my text'

    def initialize_plot(self, value):
        # method initialize the plot in the very beginning
        self.text = self.ax.text(0.5, 0.5, value, fontsize="xx-large")

    def update(self, value):
        # method to update the plot
        self.text.set_text(value)
```

assign the formatoption to a plotter

In [12]:
```
class MyPlotter(Plotter):
    my_fmt = MyFormatoption('my_fmt')
```

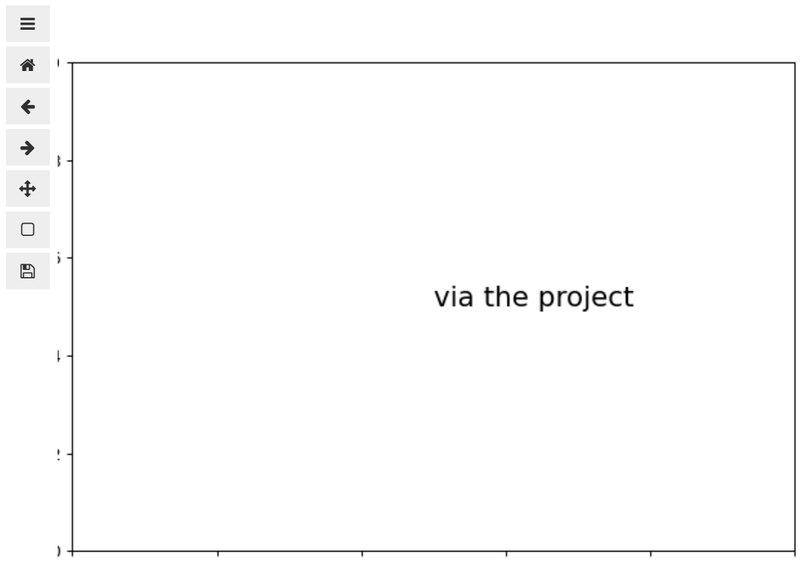
Low-level interface

In [13]:
```
# make the plot

ds = psy.open_dataset('data/demo.nc')

data = ds.psy.t2m

plotter = MyPlotter(data)
```



Low-level interface

In [13]:
```
# make the plot

ds = psy.open_dataset('data/demo.nc')

data = ds.psy.t2m

plotter = MyPlotter(data)
```

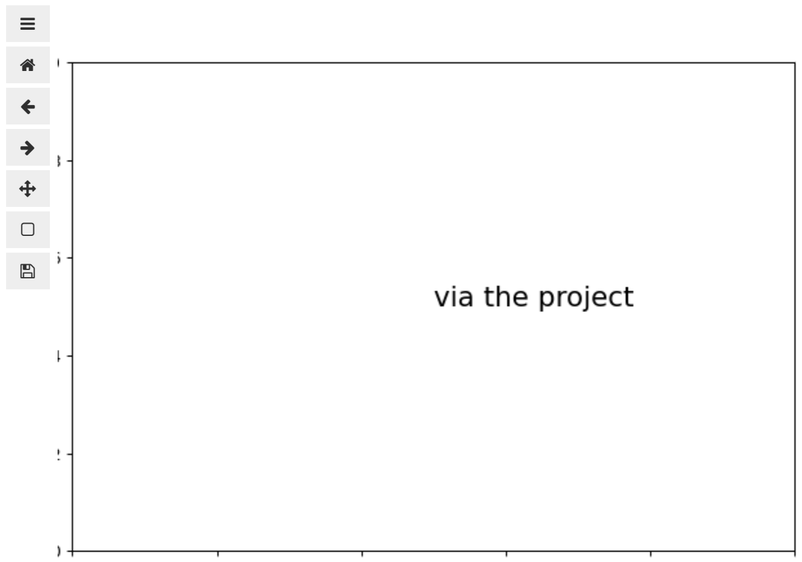


In [14]:
```
# turn it into a project

from psyplot.project import Project
project = Project([data])

project
```

Out[14]:
```
1 Main psyplot.project.Project([    arr0: 4-dim DataArray
of t2m, with (time, lev, lat, lon)=(5, 4, 96, 192), ])
```

Low-level interface

In [13]:
```
# make the plot

ds = psy.open_dataset('data/demo.nc')

data = ds.psy.t2m

plotter = MyPlotter(data)
```



In [14]:
```
# turn it into a project

from psyplot.project import Project
project = Project([data])

project
```

Out[14]:
```
1 Main psyplot.project.Project([    arr0: 4-dim DataArray
of t2m, with (time, lev, lat, lon)=(5, 4, 96, 192), ])
```

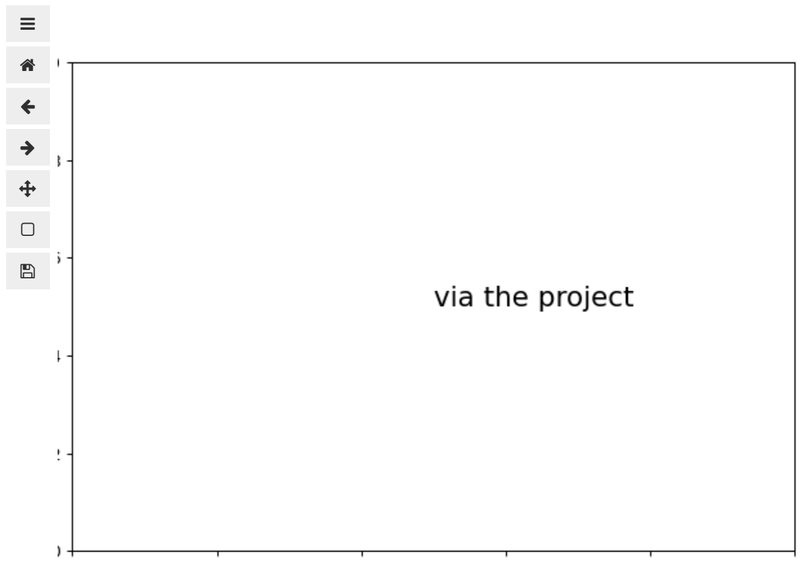
In [15]:
```
# update the plot

data.psy.update(my_fmt="via the data accessor")
```

In [16]:
```
plotter.update(my_fmt="via the plotter")
```

In [17]:
```
project.update(my_fmt="via the project")
```

Low-level interface

In [13]:
```
# make the plot

ds = psy.open_dataset('data/demo.nc')

data = ds.psy.t2m

plotter = MyPlotter(data)
```



In [14]:
```
# turn it into a project

from psyplot.project import Project
project = Project([data])

project
```

Out[14]:
```
1 Main psyplot.project.Project([    arr0: 4-dim DataArray
of t2m, with (time, lev, lat, lon)=(5, 4, 96, 192), ])
```

In [15]:
```
# update the plot

data.psy.update(my_fmt="via the data accessor")
```

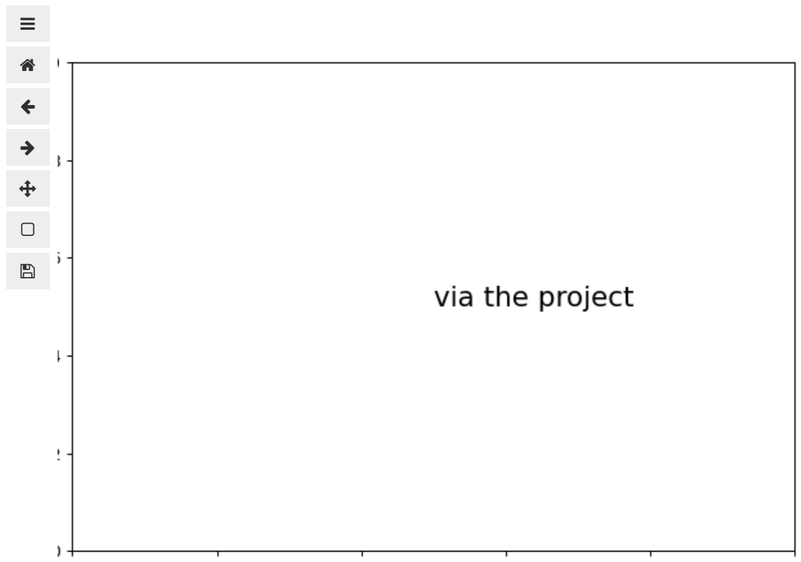
In [16]:
```
plotter.update(my_fmt="via the plotter")
```

In [17]:
```
project.update(my_fmt="via the project")
```

In [18]:
```
# finally, you should always close the project
project.close(True, True, True)
```

Low-level interface

In [13]:
```
# make the plot

ds = psy.open_dataset('data/demo.nc')

data = ds.psy.t2m

plotter = MyPlotter(data)
```



In [14]:
```
# turn it into a project

from psyplot.project import Project
project = Project([data])

project
```

Out[14]:
```
1 Main psyplot.project.Project([    arr0: 4-dim DataArray
of t2m, with (time, lev, lat, lon)=(5, 4, 96, 192), ])
```

In [15]:
```
# update the plot

data.psy.update(my_fmt="via the data accessor")
```

In [16]:
```
plotter.update(my_fmt="via the plotter")
```

In [17]:
```
project.update(my_fmt="via the project")
```

In [18]:
```
# finally, you should always close the project
project.close(True, True, True)
```

Tutorial notebook:
https://psyplot.github.io/examples/general/example_exte

High-level interface

You can register plotters such that they become available via `psyplot.project.plot.<something>`, or `xarray.Dataset.psy.plot.<something>`.

High-level interface

You can register plotters such that they become available via `psyplot.project.plot.<something>`, or `xarray.Dataset.psy.plot.<something>`.

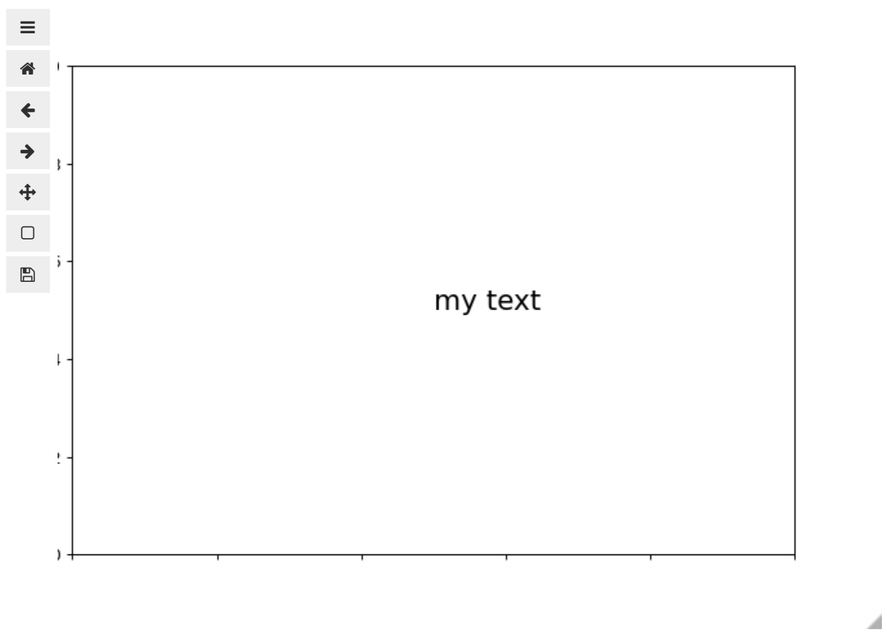
In [19]:
```
psy.register_plotter('my_plotter', MyPlotter.__module__,
                     'MyPlotter', MyPlotter)
```

High-level interface

You can register plotters such that they become available via `psyplot.project.plot.<something>`, or `xarray.Dataset.psy.plot.<something>`.

In [19]:
```
psy.register_plotter('my_plotter', MyPlotter.__module__,
                     'MyPlotter', MyPlotter)
```

In [20]:
```
psy.plot.my_plotter('data/demo.nc', name='t2m')
```



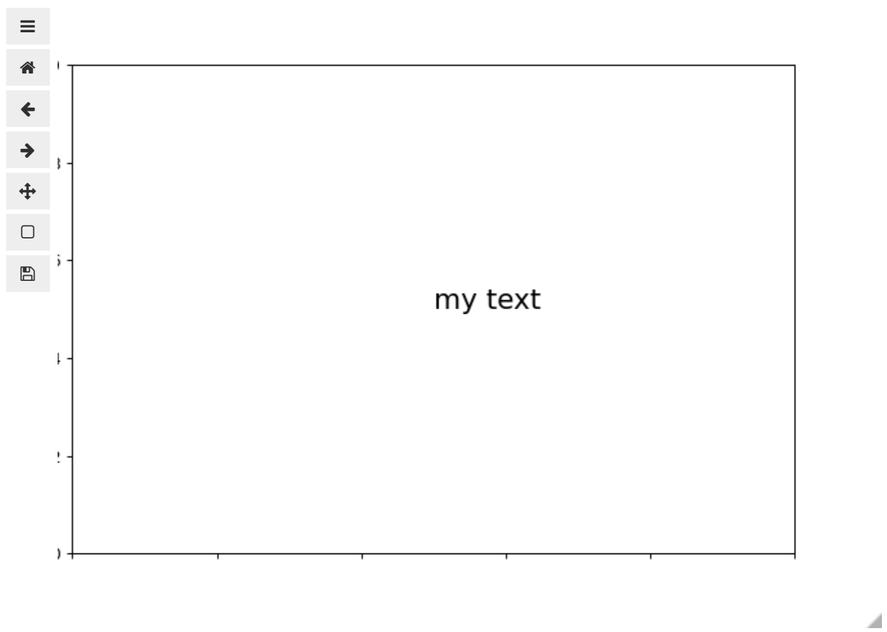
Out[20]:
```
psyplot.project.Project([    arr2: 4-dim DataArray of t2m,
with (time, lev, lat, lon)=(5, 4, 96, 192), ])
```

High-level interface

You can register plotters such that they become available via `psyplot.project.plot.<something>`, or `xarray.Dataset.psy.plot.<something>`.

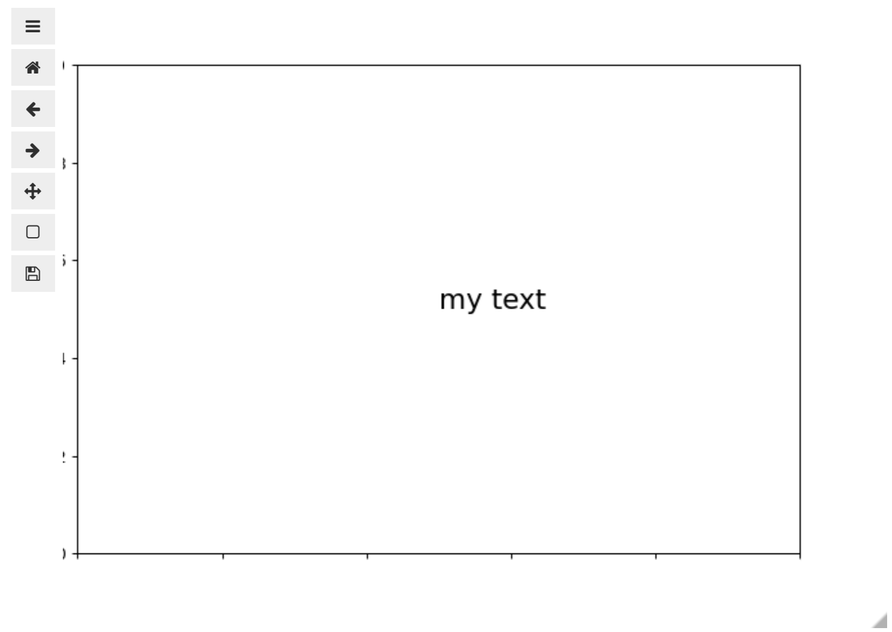
In [19]:
```
psy.register_plotter('my_plotter', MyPlotter.__module__,
                     'MyPlotter', MyPlotter)
```

In [20]:
```
psy.plot.my_plotter('data/demo.nc', name='t2m')
```



Out[20]:
```
psyplot.project.Project([    arr2: 4-dim DataArray of t2m,
with (time, lev, lat, lon)=(5, 4, 96, 192), ])
```

In [21]:
```
ds.psy.plot.my_plotter(name="t2m")
```



Out[21]:
```
psyplot.project.Project([    arr3: 4-dim DataArray of t2m,
with (time, lev, lat, lon)=(5, 4, 96, 192), ])
```

Plugins for visualization

psyplot is the core that defines the framework (Plotter, Formatoption, Project, CFDecoder), the plot methods are implemented by plugins:

- `psy-simple`: for standard 1D and 2D plot
    - e.g. lineplot, plot2d, vector, barplot
- `psy-maps`: for georeferenced plots (i.e. maps)
    - e.g. mapplot, mapvector, etc.
- `psy-reg`: for regression analysis
    - linreg, densityreg

Plugins for visualization

psyplot is the core that defines the framework (Plotter, Formatoption, Project, CFDecoder), the plot methods are implemented by plugins:

- `psy-simple`: for standard 1D and 2D plot
  - e.g. lineplot, plot2d, vector, barplot
- `psy-maps`: for georeferenced plots (i.e. maps)
  - e.g. mapplot, mapvector, etc.
- `psy-reg`: for regression analysis
  - linreg, densityreg

```
In [22]: psy.plot.show_plot_methods()
```

```
barplot
    Make a bar plot of one-dimensional data
combined
    Plot a 2D scalar field with an overlying vector field
density
    Make a density plot of point data
fldmean
    Calculate and plot the mean over x- and y-dimensions
horizontal_mapcombinedtransect
    Open and plot data via :class:`psy_transect.maps.Horiz
ontalTransectCombinedPlotter` plotters
horizontal_maptransect
    Open and plot data via :class:`psy_transect.maps.Horiz
ontalTransectFieldPlotter` plotters
horizontal_mapvectortransect
    Open and plot data via :class:`psy_transect.maps.Horiz
ontalTransectVectorPlotter` plotters
lineplot
    Make a line plot of one-dimensional data
mapcombined
    Plot a 2D scalar field with an overlying vector field
on a map
mapplot
    Plot a 2D scalar field on a map
mapvector
    Plot a 2D vector field on a map
my_plotter
    Open and plot data via :class:`__main__.MyPlotter` plo
tters
plot2d
    Make a simple plot of a 2D scalar field
vector
    Make a simple plot of a 2D vector field
vertical_maptransect
    Open and plot data via :class:`psy_transect.plotters.V
erticalMapTransectPlotter` plotters
vertical_transect
    Open and plot data via :class:`psy_transect.plotters.V
erticalTransectPlotter` plotters
violinplot
    Make a violin plot of your data
```

They already have a lot of formatoptions available

In [23]:
```
psy.plot.mapplot.keys(grouped=True)
```

```
******************
Axes formatoptions
******************
+------------+------------+------------+
| background | tight      | transpose  |
+------------+------------+------------+

*************************
Color coding formatoptions
*************************
+-------------+-------------+-------------+-------------+
| bounds      | cbar        | cbarspacing | cmap        |
+-------------+-------------+-------------+-------------+
| ctickprops  | cticksize   | ctickweight | extend      |
+-------------+-------------+-------------+-------------+
| levels      | miss_color  |             |             |
+-------------+-------------+-------------+-------------+

*******************
Label formatoptions
*******************
+----------------+----------------+----------------+----------------+
| clabel         | clabelprops    | clabelsize     | clabelweight   |
+----------------+----------------+----------------+----------------+
| figtitle       | figtitleprops  | figtitlesize   | figtitleweight |
+----------------+----------------+----------------+----------------+
| text           | title          | titleprops     | titlesize      |
+----------------+----------------+----------------+----------------+
| titleweight    |                |                |                |
+----------------+----------------+----------------+----------------+

**************************
Miscallaneous formatoptions
**************************
+----------------+----------------+----------------+----------------+
| clat           | clip           | clon           | datagrid       |
+----------------+----------------+----------------+----------------+
| grid_color     | grid_labels    | grid_labelsize | grid_settings  |
+----------------+----------------+----------------+----------------+
| interp_bounds  | lonlatbox      | lsm            | map_extent     |
+----------------+----------------+----------------+----------------+
| mask_datagrid  | projection     | stock_img      | transform      |
+----------------+----------------+----------------+----------------+
| xgrid          | ygrid          |                |                |
+----------------+----------------+----------------+----------------+

***********************
Axis tick formatoptions
***********************
```

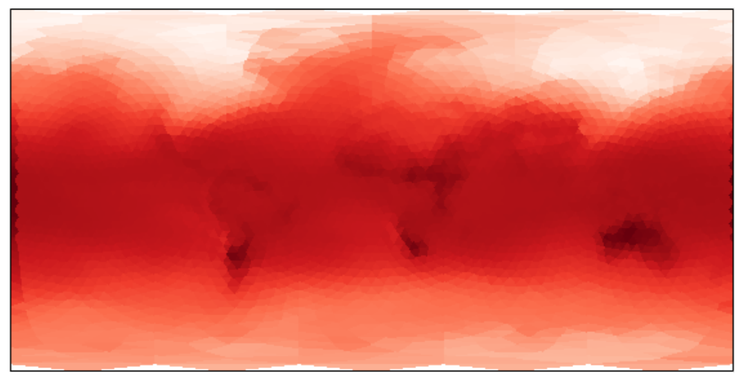
Make your own data plotter

In [24]:
```python
import numpy as np
from psy_simple.plotters import CMap, Bounds
from psy_maps.plotters import Transform, MapPlot2D

from psyplot.plotter import Plotter

class MySimpleMapplot(Plotter):

    # Specify the defaults
    rc = {
        "cmap": "Reds",
        "norm": None,
        "transform": "cf",
        "plot": "poly",
    }

    def convert_coordinate(self, coord, *variables):
        if coord.attrs.get("units", "").startswith("radian") or any(
            var.attrs.get('units', '').startswith('radian')
            for var in variables
        ):
            coord = coord.copy(data=coord * 180. / np.pi)
            coord.attrs["units"] = "degrees"
        return coord

    # Specify the formatoptions
    cmap = CMap("cmap", bounds="norm")
    norm = Bounds("norm")
    transform = Transform("transform")

    plot = MapPlot2D("plot", bounds="norm")


def plot(data, ax, **formatoptions):
    return MySimpleMapplot(
        data, ax=ax, **formatoptions
    )
```

Make your own data plotter

In [24]:
```python
import numpy as np
from psy_simple.plotters import CMap, Bounds
from psy_maps.plotters import Transform, MapPlot2D

from psyplot.plotter import Plotter

class MySimpleMapplot(Plotter):

    # Specify the defaults
    rc = {
        "cmap": "Reds",
        "norm": None,
        "transform": "cf",
        "plot": "poly",
    }

    def convert_coordinate(self, coord, *variables):
        if coord.attrs.get("units", "").startswith("radian") or all(
            var.attrs.get('units', '').startswith('radian')
            for var in variables
        ):
            coord = coord.copy(data=coord * 180. / np.pi)
            coord.attrs["units"] = "degrees"
        return coord

    # Specify the formatoptions
    cmap = CMap("cmap", bounds="norm")
    norm = Bounds("norm")
    transform = Transform("transform")

    plot = MapPlot2D("plot", bounds="norm")


def plot(data, ax, **formatoptions):
    return MySimpleMapplot(
        data, ax=ax, **formatoptions
    )
```

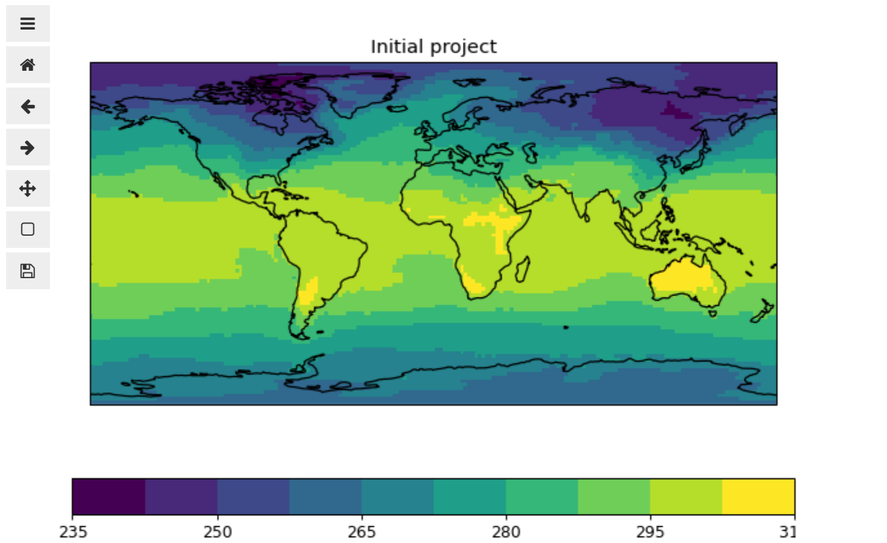
In [25]:
```python
fig, ax = plt.subplots(subplot_kw=dict(
    projection=ccrs.PlateCarree())
)
ax.set_global()

ds = psy.open_dataset("data/icon.nc")

plot(ds.psy.t2m, ax=ax);
```



Other features of psyplot

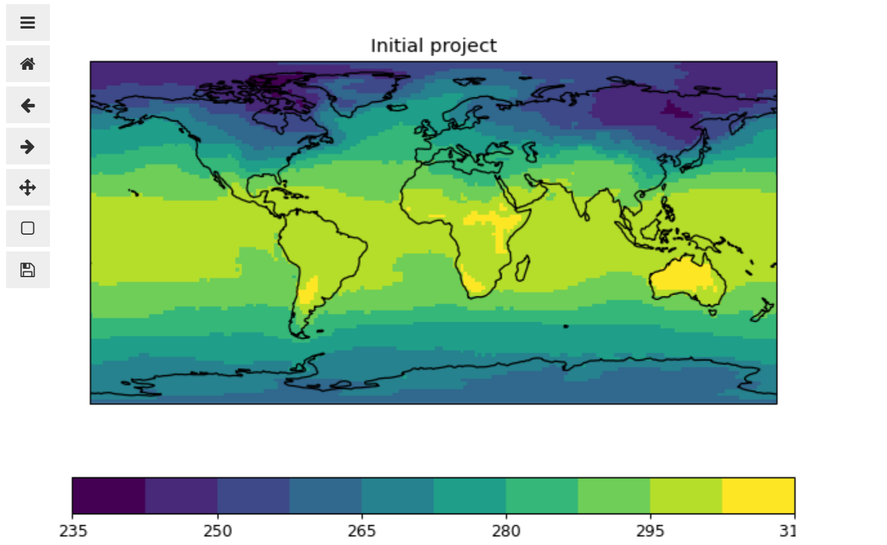
Save and load projects

In [26]:
```
initial = psy.plot.mapplot('data/demo.nc', name='t2m', title
initial.save_project('my-project.pkl')
```

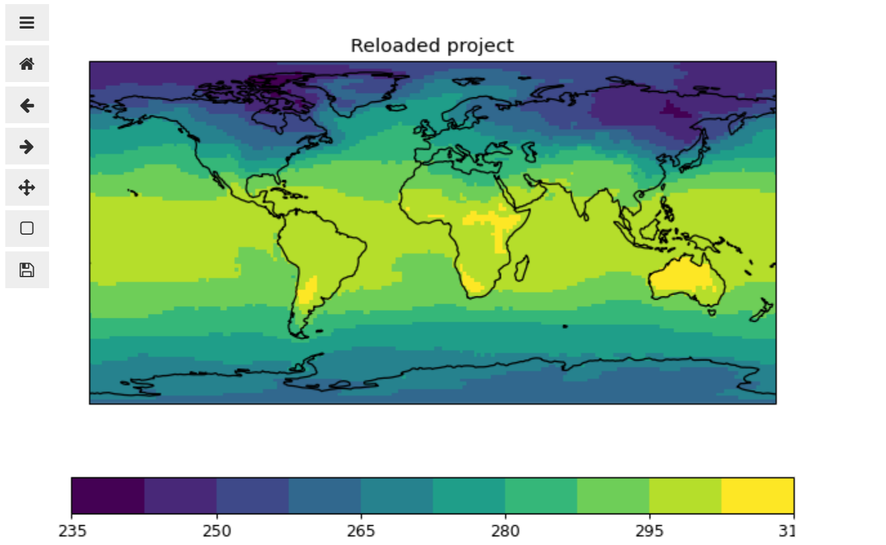


Save and load projects

In [26]:
```
initial = psy.plot.mapplot('data/demo.nc', name='t2m', title
initial.save_project('my-project.pkl')
```



In [27]:
```
reloaded = psy.Project.load_project('my-project.pkl')
reloaded.update(title='Reloaded project')
```



Export plots

In [28]:
```
plt.ioff()

with psy.plot.mapplot('data/demo.nc', name='t2m', time=[0, 1, 2], title='%(time)s') as sp:
    sp.export('data/step-%i.png')

plt.ion()

!ls data/step-?.png
```

```
data/step-1.png  data/step-2.png  data/step-3.png
```

Export plots

In [28]:
```
plt.ioff()

with psy.plot.mapplot('data/demo.nc', name='t2m', time=[0, 1, 2], title='%(time)s') as sp:
    sp.export('data/step-%i.png')

plt.ion()

!ls data/step-?.png
```

```
data/step-1.png  data/step-2.png  data/step-3.png
```

In [29]:
```
from IPython.display import display, HTML, Image
s = '<table><tr><td><img src="data/step-1.png"></td><td><img src="data/step-2.png"></td><td><img src="data/step-3.png"></td></tr></tab
display(HTML(s))
```

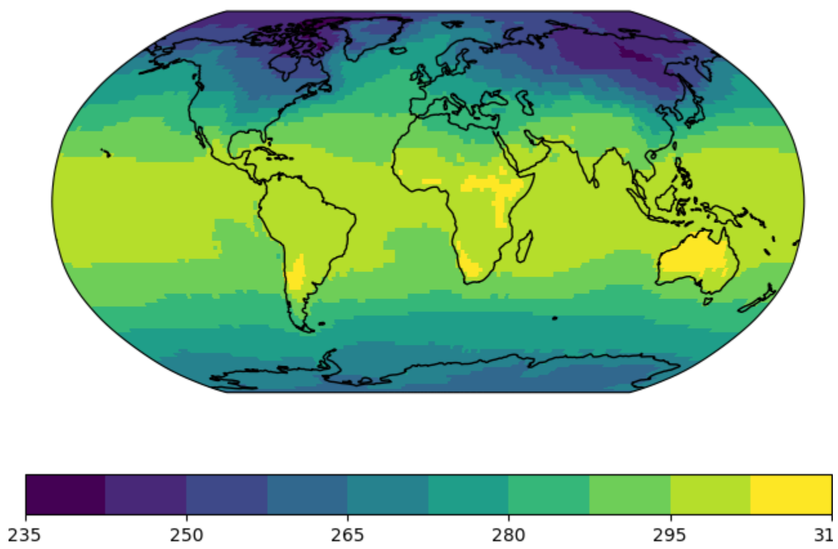
Generate plots from the command line

In [30]:
```
!echo 'projection: robin' > fmt.yml
!psyplot data/demo.nc -n t2m -pm mapplot -fmt fmt.yml -o data/output.png
```

Generate plots from the command line

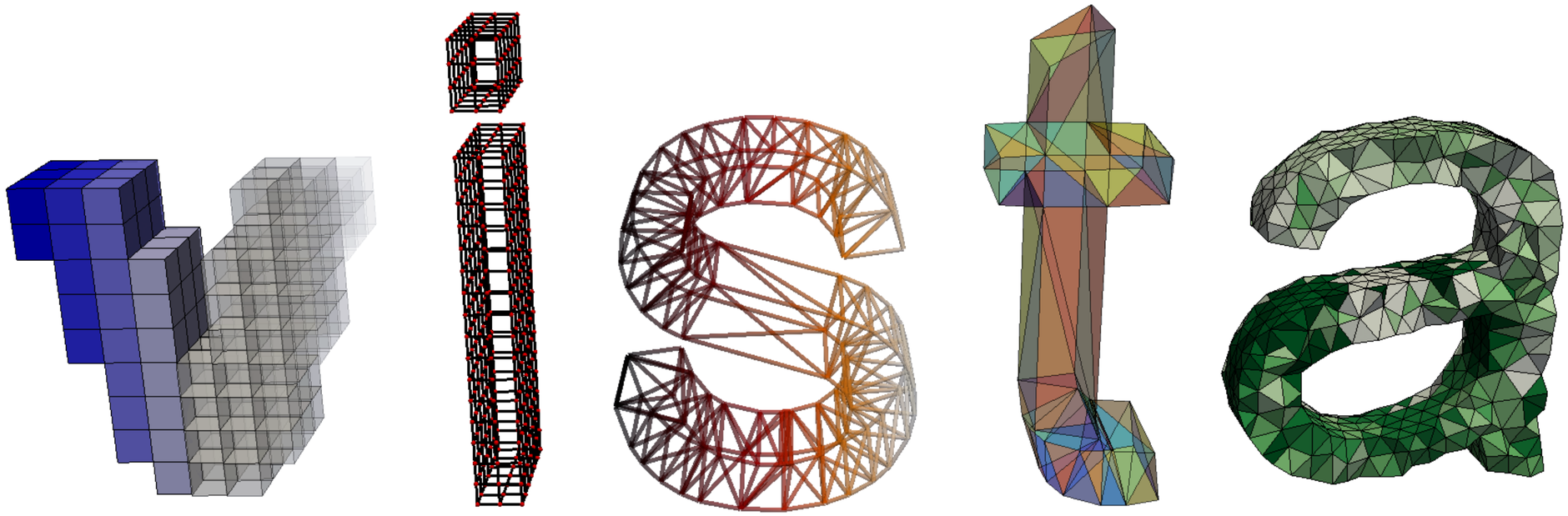
In [30]:
```
!echo 'projection: robin' > fmt.yml
!psyplot data/demo.nc -n t2m -pm mapplot -fmt fmt.yml -o data/output.png
```

In [31]:
```
display(Image('data/output.png'))
```



Outlook

3D visualization (not matplotlib, VTK)



pyvista: VTK and matplotlib

In [32]:
```
import psy_vtk.plotters as pvtk

ds = psy.open_dataset('data/rectilinear.nc')

globe_plot = pvtk.GlobePlotter(ds.psy.t2m)
disp = globe_plot.ax.show()
```

pyvista combines the power and efficiency of the (pretty difficult) VTK python bindings with a well-documented interface and matplotlib-like functionalities.

And it can even work with psyplot pretty much out of the box!

pyvista: VTK and matplotlib

In [32]:
```
import psy_vtk.plotters as pvtk

ds = psy.open_dataset('data/rectilinear.nc')

globe_plot = pvtk.GlobePlotter(ds.psy.t2m)
disp = globe_plot.ax.show()
```

pyvista combines the power and efficiency of the (pretty difficult) VTK python bindings with a well-documented interface and matplotlib-like functionalities.

And it can even work with psyplot pretty much out of the box!

In [33]:
```
globe_plot.update(cmap='Reds')
```

In [34]:
```
globe_plot.update(datagrid=True)
```

Immediate support of all grids

In [35]:
```
ds = psy.open_dataset('data/rectilinear.nc')

globe_plot = pvtk.GlobePlotter(ds.psy.t2m)
globe_plot.ax.show();
```

In [36]:
```
ds = psy.open_dataset('data/curvilinear.nc')

globe_plot = pvtk.GlobePlotter(ds.psy.t2m)
globe_plot.ax.show();
```

Visualizing big data

For 4.4 million cells:

psy-vtk needs a couple of seconds to visualize it and it works fluently.

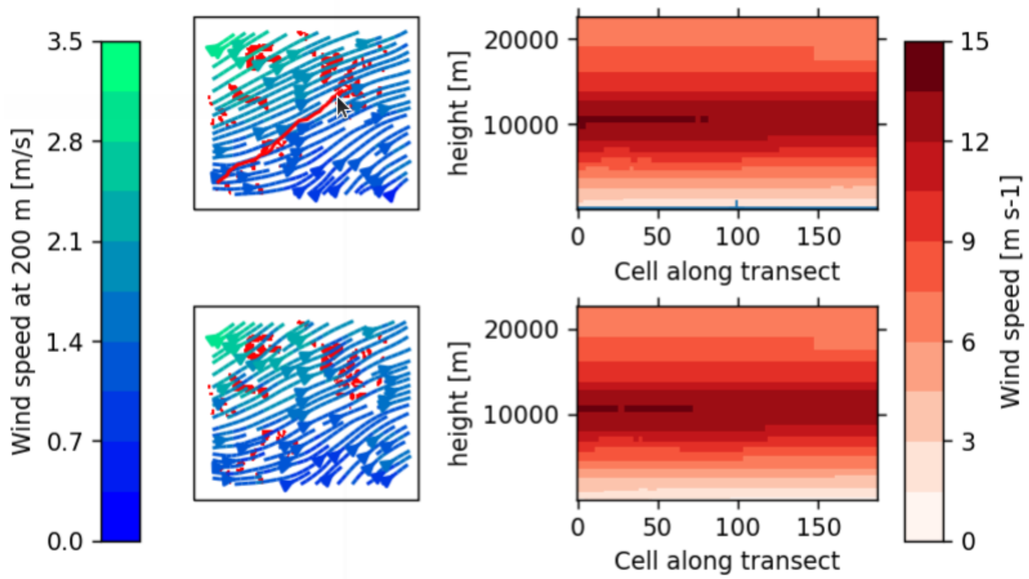
```
In [37]: Video("data/screencast.mp4", embed=True, html_attributes="au
```

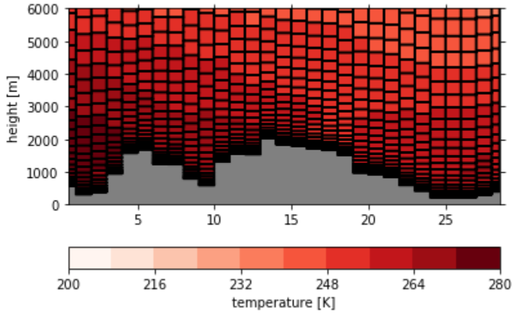
Out[37]:

Transects

psy-transect

- interpolate rastered (or unstructured) data on to a path.
- display vertical profiles (with respect to orography, if available)
- supports vector (wind) or scalar fields (temperature)
- made for interactive usage





Summary

The framework

- the psyplot core for the data model, and plugins for various visualizations
- designed to be flexible and sustainable
- equipped via flexible graphical user interface

Summary

The framework

- the psyplot core for the data model, and plugins for various visualizations
- designed to be flexible and sustainable
- equipped via flexible graphical user interface

The data model

- based on a netCDF-like infrastructure and interpretes CF- and UGRID conventions
- support for multiple grids: rectilinear, circumpolar and unstructured

Summary

The framework

- the psyplot core for the data model, and plugins for various visualizations
- designed to be flexible and sustainable
- equipped via flexible graphical user interface

The data model

- based on a netCDF-like infrastructure and interpretes CF- and UGRID conventions
- support for multiple grids: rectilinear, circumpolar and unstructured

Scriptability

- close to the data with a minimum of visualization overhead (compared to Paraview or something else)
- can easily be enhanced by other powerful libraries, such as scipy, scikit-learn, etc.