nc2map User Manual

Module for an interactive plotting of NetCDF files with Python, version 0.0b

Philipp Sommer

July 1, 2015

Contents

Preface

This user manual shall serve as an introduction to the usage of the `nc2map` Python module to plot NetCDF files. Please note that this is only an introduction which shall guide you to the principal structure of the module. To see the full documentation of the methods and so one, use the Python built-in **help** function (e.g. **help**`(nc2map.Maps.make_movie)`).
In case of any problems, do not hesitate to contact the author:
`philipp.sommer@studium.uni-hamburg.de`.

1 Introduction

Visualizing data is a major part of the scientific work, not only for publications but also to analyse ones own data. However commonly large scripts or more or less complex programmes are necessary to visualize your data especially when it comes to two-dimensional global maps. Therefore this module has been developed with a special focus on the visualization of NetCDF files, a commonly used data format in climate sciences (at least when it comes to global models), to efficiently visualize the data. They major advantages compared to other programmes and modules are:

1. with only a small number of commands you can take a detailed look into your data and make nicely looking maps (for publications or just to get an idea of your data)

2. you can easily access the data, make calculations with it and implement everything into your own plotting and evaluation routines since there are very weak dependencies

3. it is (hopefully) well documented

4. it is open source

The final goal is to also develop a graphical user interface (GUI) for the module to create something like `ncview` just better. However, since the module is very new, there is currently only support for the command line application, i.e. for the use in scripts or with `python`, `ipython`, etc.

2 Getting started

This chapter is an introduction to the `nc2map` Python module. The first section (section 2.1) deals with the general module structure, the second is a quick start showing how you can create your own maps (section 2.2) and the third is a more detailed description of the possibilities (section 2.3).

Please also find demo scripts in the `demo` directory of your `nc2map` source files.

2.1 A note about the module structure

As comparable with `matplotlibs` axes (subplot) - figure structure, `nc2map` consists of a basic class responsible for the plot (subclasses of `MapBase`) and a coordinating class: `MapsManager`. Each `MapBase` instance thereby controls exactly one axes. A `MapsManager` instance on the other hand controls multiple `MapBase` instances.

The most important `MapsManager` subclass is the `Maps` class which is not only designed to control many `MapBase` instances but also colorbars, evaluations, output and updates to make everything interactive. Hence, the thing you probably will deal most with are instances of the `Maps` class.

Furthermore, you will probably not deal with the `MapBase` class, but rather with its subclasses. Those are `FieldPlot`, to plot a two dimensional scalar field (e.g. temperature, see the upper row in figure 2.1) and `WindPlot`, to visualize flows (e.g. wind or ocean currents, see the lower row in figure 2.1).

2.2 Quick start

The simplest way how to plot open a NetCDF file and visualize it is with the `Maps` class. If `ncfile` is the variable containing the path to your NetCDF file, you can visualize the variables with

Listing 2.1: Basic initialization of a `Maps` instance

```
import nc2map
ncfile = 'my-own-netcdf-file.nc'
mymaps = nc2map.Maps(ncfile)      # initialize Maps instance
mymaps.show()                     # show all figures
```

Or select specific variables via their name in the NetCDF file, e.g.

```
mymaps = nc2map.Maps(ncfile, vlst=['t2m', 'u'])
```

To visualize wind vectors, you can use the `u` and `v` keywords, e.g.

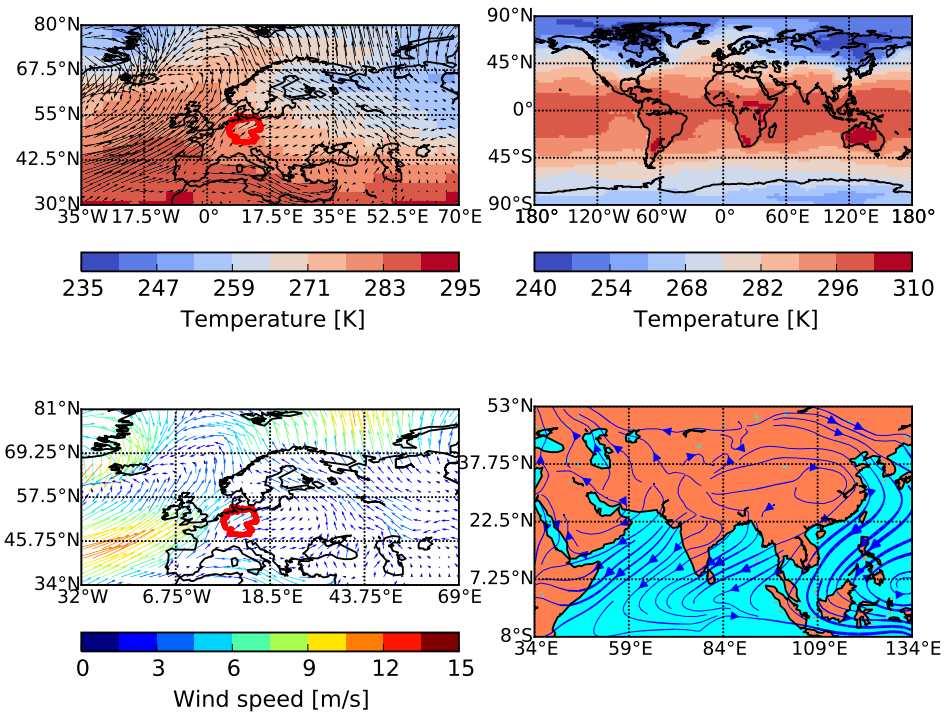

Figure 2.1: Demonstration of the different plot types with Pythons nc2map package

```
mymaps = nc2map.Maps(ncfile, u='u', v='v')
```

or if want to visualize only vector data without an underlying scalar field (e.g. temperature), you can do this by

```
mymaps = nc2map.Maps(ncfile, u='u', v='v', windonly=True)
```

You can then use the `update` method to update the plot via formatoptions (see next chapter 5). For example we can update the title of all plots via

```
mymaps.update(title='My test title of variable %(var)s.')
```

Here `%(var)s` is replaced by the name of the specific variable that is shown in each plot, as it is used in the NetCDF file (actually you can use any meta data from the variables in the NetCDF file, see section 2.6). You can also update only specific `MapBase` instances by using any of the meta attributes (e.g. `time`, or `level`, or `var`) or directly via the instance specific attribute `name`.

Hence let's say we only want to update the temperature (stored in variable `t2m`) and zonal wind (stored in variable `u`). Then we can update these `MapBase` instances via

```
mymaps.update(title='My test title of variable %(var)s.',
              vlst=['t2m', 'u'])
```

For a detailed usage of the `update` method please refer to the python built-in **help** function.

Another possibility (see also next section 2.3) would be to include the formatoptions directly in the initialization by the use of the `fmt` keyword via a dictionary

```
import nc2map
ncfile = 'my-own-netcdf-file.nc'
fmt = {'title': 'My test title of variable %(var)s.'}
mymaps = nc2map.Maps(ncfile, fmt=fmt)  # initialize Maps instance
mymaps.show()                          # show all figures
```

There are some demo scripts in the demo directory of your nc2map distribution which may show you some possible applications.

2.3 Initializing a `nc2map.Maps` instance

2.3.1 How to specify the NetCDF files

`nc2map` is primarily designed to plot NetCDF files, however it might be extended in the future (e.g. for GeoTiff files). You can simply pass in one single NetCDF file, use wildcards (e.g. * or ?) or a list of NetCDF files.

One single file: Simply use the file name, e.g.

```
mymaps = nc2map.Maps('my-netcdf-file.nc')|
```

Using wild cards: Same as with a single file name, e.g.

```
mymaps = nc2map.Maps('my-*-file.nc')
```

Using multiple files: Simply use a list of file names, e.g.

```
mymaps = nc2map.Maps(['my-netcdf-file1.nc', 'my-netcdf-file2.
   nc'])
```

By default, `nc2map.Maps` uses the `netCDF4.Dataset` to visualize a single NetCDF file and the `netCDF4.MFDataset` to visualize multiple NetCDF files. You can set manually which of the above classes is used via the `mode` key word (`mode='NCReader'` of `mode='MFNCReader`) in the initialization of a `Maps` instance.
Furthermore, instead of passing in the NetCDF file (or a list of NetCDF files), you can initialize the `Maps` instance via setting the `ncfile` equal to a dictionary, e.g.

```
mymaps = nc2map.Maps({'filename': 'my-netcdf-file.nc', 'mode': 'a'})
```

to open the NetCDF file in an editor mode. The key-value pairs of the dictionary are then assumed to represent the keyword arguments used for the initialization of the `nc2map.readers.ArrayReader` instance.
Finally you can also set the `ncfile` keyword with an already existing `reader` instance.

2.3.2 Specifying what to plot via dimension identifiers

The Maps class takes a bunch of possible keywords for the initialization (see help(nc2map.Maps.__init__) method for details). We already saw in listing 2.1, how to generally visualize a NetCDF file. However those commands would show all variables at the first time step, level, etc., which is maybe not desired. Therefore we can use the dimensions in the NetCDF file to be more specific. Which dimensions there are, depends on your specific NetCDF file. During the initialization of a Maps instance, you can set any dimension you want, e.g.

```
# variables 't2m' and 'u' at the 2nd time step for 1st and 2nd level
mymaps = nc2map.Maps(ncfile, vlst=['t2m', 'u'], time=1, level=[0, 1])
```

This will open 4 maps in total and assign automatically generated names mapo0, mapo1, mapo2 and mapo3 to the generated MapBase instances. Note that for any dimension you specify in this way that is iterable[1], one MapBase instance is created. Hence,

```
mymaps = nc2map.Maps(ncfile, vlst=['t2m', 'u'], time=range(5), level
    =[0, 1])
```

would create 20 maps in total (not recommended because far too much!).

Therefore, you can be more specific, by using the names keyword in the initialization of the Maps instance and a dictionary. This might look like

```
names = {'mymap1': {'time': 1},
         'mymap2': {'time': 1, 'level': 1}}
  mymaps = nc2map.Maps(ncfile, names=names)
```

and opens exactly two maps, one with time=1 and level=0 and one with time=1 and level=1. This may also be useful if your variables in the NetCDF file have different dimensions.

If you do not use the names keyword as a dictionary but instead give another iterable (or a string with '{0}' in it, e.g. mymodel{0}), those will be used as names for the MapBase instances, where the '{0}' will be replaced by a counter. However, maybe it is not so useful to define your own names, but rather to give some meta attributes directly to the MapBase via the meta key. As an example

Listing 2.2: Assigning your own meta informations

```
mymaps = nc2map.Maps(ncfile, meta={'model': 'my first model'})
mymaps.addmap(ncfile, meta={'model': 'my second model'})
```

will allow you to later address the MapBase instances you want via the model key, e.g.

```
model_maps1 = mymaps.get_maps(model='my first model')
model_maps2 = mymaps.get_maps(model='my second model')
```

Instead of setting time to 1, you can also use the time information in the NetCDF file (see section 3.1)

[1] iterables in python are everything that does not raise an Error when using the built-in iter function. The most prominent example are lists (e.g. [1, 2, 3], or range(5) or xrange(5)).

Those dimensions can also be changed via the `Maps.update` method. For example

```
mymaps.update(fmt={'time': 1}, time=0)
```

will update all maps that currently show the first time step to the second. For time however, you can also use the `nextt` and `prevt` methods.

The function that is used in the initialization to add new maps to the `Maps` instance, is the `addmap` method. Hence, to add another map to the `Maps` instance, you can simply use

```
   mymaps.addmap(ncfile, ...)
```

Furthermore there is the possibility to make one dimensional line plots with data from the NetCDF file. The syntax is somewhat similar, e.g.:

```
mymaps = nc2map.Maps(ncfile, vlst=['t2m', 'u'], level=[0, 1], lat=0,
   lon=0, linesonly=True)
```

will draw lines over all time steps in the NetCDF file into one subplot for the first and second level. The corresponding method that is used is the `addline` method.

2.3.3 Specifying how to plot via formatoption keywords

There exist over 60 formatoptions, that can be used to exactly control the appearance of your plot. Each formatoption can be set with one single keyword (see next chapter 5).

As stated already in section 2.2, you can give formatoptions directly to the initialization of the `Maps` instance without using the `update` method. These can be done via simply setting up a dictionary with formatoption keywords like

```
fmt = {'title': 'my test title',
       'cmap': 'RdBu'}
```

However this would set the title for all variables, all time steps and all levels, in other words for all `MapBase` instances that are controlled by the `Maps` instance. But sometimes we do not want the same formatoptions for each `MapBase` instance. For example we maybe want a colormap going from red to blue for precipitation (e.g. `pyplots 'RdBu_r'` colormap) and a colormap going from blue to red for temperature (e.g. `pyplots 'RdBu_r'` colormap). Therefore we can set up the formatoptions dictionary more specifically. In our case for example let's assume that precipitation is stored in the variable `precip` and temperature in the variable `t2m`. Our formatoption dictionary for the initialization will then be

```
fmt = {'title': ' %(long_name)s',
       't2m': {'cmap': 'RdBu_r'},
       'precip': {'cmap': 'RdBu'}
      }
```

i.e. we used the variable identifier as a key in the `fmt` dictionary whose value is another `fmt` dictionary. Please note that the `'title'` formatoption is set outside of the variable specific dictionaries which means that this is used as a default value for all `MapBase`

Listing 2.3: Example of how to define a nested formatoptions dictionary.

```
default_title = 'Default title'
mapo0_title = 'Title used for mapo0'
t2m_title = 'Default title for t2m variable'
t2m_t2_title = 'Title for third time step of t2m variable'
t1_title = 'Default title for second time step'

fmt = {'title': default_title,
       'mapo0': {'title': mapo0_title},
       't2m': {'title': t2m_title,
               't2': {'title': t2m_t2_title}},
       't1': {'title': t1_title}
       }
```

instances in the `Maps` instance. However, since we refer here to the `long_name` attribute inside the NetCDF file, the titles will not be the same. Hence setting up the above dictionary like

```
fmt = {'title': 'my test title',
       't2m': {'cmap': 'RdBu_r',
               'title': 'Another title'},
       'precip': {'cmap': 'RdBu'}
      }
```

would result in a title being "Another title" for all `t2m` `MapBase` instances and "my test title" for all the others (e.g. `precip`).

This syntax does not only work for variables but also slightly modified for `times`, `levels` and `names`. The `time` step furthermore has to come with a leading `t` followed by the time step as string, whereas the `level` has to come with a leading `l` followed by the vertical step as string. Hence

```
fmt = {'t0': {'title': 'This applies only for the 0-th time step'}}
```

will only modify the titles of `MapBase` instances with `time == 0` and

```
fmt = {'l0': {'title': 'This applies only for the 0-th level'}}
```

will only modify the titles of `MapBase` instances with `level == 0`. Finally

```
fmt = {'mapo0': {'title': 'This applies only for the 0-th level'}}
```

will only modify the title of the `MapBase` instance with the name `mapo0`.

Furthermore the dictionaries can be nested, where the order of hierarchy is

$$\texttt{name} \succ \texttt{var} \succ \texttt{time} \succ \texttt{level}.$$

As an example look into listing 2.3. Here we set the title for the `MapBase` instance with `name == mapo0` to `mapo0_title`, the title for all `MapBase` instances with `time == 1` and `var != 't2m'` to `t1_title`, the title for all time step and levels with `var == 't2m'` and `time != 2` to `t2m_title` and finally for `var == 't2m'` and `time == 2` to `t2m_t2_title`.

2.4 Output methods

There are several methods of the `Maps` class to save what you created:

`output`: Save all (or some) figures into one (or more) `pdf` files, `pngs`, `jpgs`, etc.

`make_movie`: Make a movie (e.g. `.gif` or `.mp4`) of your `MapBase` instances.

`save`: Creates a python script that initializes the `Maps` instance in its current state (including all`MapBase` and `LinePlot` instances and their formatoptions.) This file can then be loaded by the `load` function to restore the `Maps` instance.

2.5 Helper functions accessing the documentation

There are several helper functions to cope with the visualization of your data:

`show_colormaps`: This function can be used to show all the colormaps that are predefined by the `nc2map` module and `pyplot`. You can also use it to visualize your own created colormap to see, how it will finally look like.

`show_fmtkeys`: This function shows the possible formatoption keywords or checks whether the the keywords specified by the user are possible.

`show_fmtdocs`: This function shows the possible formatoption keywords or checks whether the the keywords specified by the user are possible, plus their documentation.

`get_fmtkeys`: This function returns a list of possible formatoption keywords as strings, or checks whether the the keywords specified by the user are possible.

`get_fmtdocs`: This function returns a dictionary with the possible formatoption keywords as keys and their documentation as value.

`get_fnames`: Shows the possible field names in the default shape file for the `lineshapes` keyword (you can also use other shape files, see the corresponding function in the `shp_utils` function).

`get_unique_vals`: Shows the values in the default shape file for the `lineshapes` keyword (you can also use other shape files, see the corresponding function in the `shp_utils` function).

2.6 Automatical labelling

One very useful feature of the `nc2map` module is that you can very easily implement the meta data information of the NetCDF file in your plot. There are some labels (e.g. clabel, title, text, etc., see table 5.1) which you can set with strings that are then displayed on the plot. The following subsections describe how to use the text-like meta information of the NetCDF file in your labels (subsection 2.6.1) and how to display the time information (subsection 2.6.2).

2.6.1 Using text meta data

For example let's take the variable `t2m`. Of course you could simply set `title='Plot of t2m'` but why should you double your work if that variable is stored in the NetCDF file anyway? Therefore a much easier solution is setting `title='Plot of %(var)s'`. In fact in all labels you can replace any meta information stored (e.g. `long_name`) by `%(long_name)s`.

Common meta data keys are:

`var`: Name of the variable as stored in the NetCDF file (e.g. `t2m`)[2]

`standard_name`: Standard name of the variable (e.g. `evapotranspiration` for variable `evspsbl`)

`long_name`: Long name of the variable (e.g. `Temperature`)

`units`: Unit of the variable (e.g. `deg C`)

`time`: Time step

`level`: Number of the vertical level

`name`: Name of the `MapBase` instance[2]

Anyway, which meta data key is stored in a NetCDF file is determined by the NetCDF file (i.e. by the conscientiousness of the creator of the file). You can access the meta data keys via the `MapBase.meta` property of the specific `MapBase` instance. E.g. if you want to see the meta information of the variable `t2m`, simply use

```
mymaps = nc2map.Maps('my-own-netcdf-file.nc', vlst='t2m')
print mymaps.get_maps(vlst='t2m')[0].meta
```

Another possibility is to use the `get_label_dict` method of the `Maps` instance via

```
mymaps = nc2map.Maps('my-own-netcdf-file.nc', vlst='t2m')
print mymaps.get_label_dict(*mymaps.get_maps(vlst='t2m'))
```

or the `MapsManager.meta` see property.

2.6.2 Displaying the time

As we saw in the previous subsection 2.6.1, one possibilty to display the time step is via `%(time)s` in our labels. However, one of the great advantages of NetCDF files is that they (almost always) follow the CF-conventions[3] and store the time data in relative time units (e.g. `days since 1992-10-8 15:15:42.5 -6:00`) or at least in absolute time units (`day as %Y%m%d.%f`). `nc2map` interpretes those two units and uses the python `datetime` module to display the information in the labels. Hence to display the time of your `MapBase` instance,

[2] This information is always stored in the `MapBase` instance, it is not a meta data information in the NetCDF file

[3] `http://cfconventions.org/`

you can use all format strings suitable with the `strftime` function of the `datetime.datetime` module[4]. For example lets assume that my `MapBase` instance shows the variable `t2m` on the 2nd of March, 2015. Then setting my title as `title='%(var)s in %B, %Y'` would result in the title `'t2m in March, 2015'` (or `'t2m in März, 2015'` if you are in Germany).

[4]For a list of format strings see `https://docs.python.org/2/library/datetime.html`

3 NetCDF restrictions and dimension handling

The `nc2map` module is designed to very flexible. Therefore in principle every NetCDF file can be read. You can have as many dimensions in your NetCDF file as you want and you can name them as you want. However, only one can always be regarded as one of the special dimensions latitude, longitude, time and level. In detail:

1. Your NetCDF file should only have one longitude variable and one latitude variable and the data of this dimensions have to be stored in two different variables
2. Only one variable can be considered as the variable for the time dimension
3. Only one variable can be considered as the variable for the level dimension

You can tell the `reader` at the initalization, what the `levelnames` are it shall look for, the `timenames`, the `lonnames` and the `latnames`. Those keywords can also be set at the initialization of a `Maps` instance. (see **help**`(nc2map.Maps)` and **help**`(nc2map.readers.ReaderBase)`).

3.1 Using the time information in the NetCDF file

Concerning the time dimension, it is recommended to use relative or absolute time units. If the time information in the `reader` (i.e. NetCDF file) is stored in relative (e.g. hours since ...) or absolute (day as %Y%m%d.f) units, strings like `%Y` for year or `%m` for the month as given by the python datetime package in labels like title, text, etc., are also replaced by the specific time information. Furthermore you can then select the time step not only via the time step explicitly (i.e. the integer), but by the time information. You can then either use a string in isoformat, e.g. `'1979-02'` for February 1979 or `'1979-02-01T18:45'` for February 1st, 1979 at 18:45 in the evening, a `numpy.datetime64` instance or a `datetime.datetime` instance.

4 Interactive usage

One, or maybe the most important feature of the `nc2map` module is it's capability for an interactive usage. You do not have to run the same script again and again but can use `python` or `ipython` (or any other python shell) to modify your plots at run time. I recommend to use the interactive python shell `ipython`, because it also has the `%save` magic to save your commands as a script.

However, the updating process is rather simple via the `Maps.update` method. You can also find a demo script in the `nc2map/demo` directory, but you learn it the best, if you simply try it by yourself. The `update` method takes every formatoption keyword (see next chapter 5) as keyword and any meta attribute in your `MapBase` instance as a selector. For example

Listing 4.1: Update formatoptions variable specific

```
mymaps.update(var='t2m', cmap='RdBu_r')
```

will update the colormap of all `MapBase` instances showing the NetCDF variable `t2m`. The same works for own created meta data, e.g. coming back to listing 2.2,

```
mymaps.update(model='my first model', lonlatbox='Europe')
```

will update the plot of all `MapBase` instances that were created with the `'my first model'` flag to focus on Europe, whereas all the others keep untouched. The same works for meta informations stored in the variable of the NetCDF file, e.g.

```
mymaps.update(long_name='Temperature', cmap='RdBu_r')
```

will have the same effect as listing 4.1. You can also pass in a list of meta attributes instead of strings, e.g

```
mymaps.update(model=['my first model', 'my second model'], cmap='RdBu
    ')
```

5 Formatoptions

The basic control feature for the appearance of the plot are the formatoption keywords, usually identified by the `fmt` key, for example in the initialization of a `Maps` instance or in the `Maps.update` method[1]. The possible formatoptions depend on what you are plotting. A `FieldPlot` (upper row in figure 2.1) for example has the possible formatoptions shown in table 5.1. A `WindPlot` (lower row in figure 2.1) additionally has the keywords shown in table 5.2 and simple plots (`LinePlot` instances) have the formatoptions shown in table 5.3.

The following sections are automatically generated from the `nc2map` module. They show the documentation of the formatoption keys, however there are some helper functions to display possible keys and their usage inside python: `nc2map.show_fmtkeys` and `nc2map.show_fmtdocs` (see section 2.5).[2]

5.1 Basemap and axes formatoptions

figtitlesize: string or float (Default: 12). Defines the size of the subtitle of the figure (see fontsize for possible values). This is the title of this specific axes! For the title of the figure see figtitlesize

text: String, tuple or list of tuples (x,y,s[,coord.-system][,options]]) (Default: []).

[1] However, you can either give the formatoptions directly as keyword argument to the `update` method or as dictionary via the `fmt` key.

[2] Hint: `show_fmtkeys` can also be used to look how to exactly spell the formatoption keyword you want to use.

Table 5.1: List of formatoption keywords

axiscolor	bounds	clabel	cmap	countries
cticksize	ctickweight	enable	extend	figtitle
figtitlesize	figtitleweight	fontsize	fontweight	grid
labelsize	labelweight	land_color	latlon	lineshapes
lonlatbox	lsm	mask	maskbetween	maskgeq
maskgreater	maskleq	maskless	meridionals	merilabelpos
ocean_color	paralabelpos	parallels	plotcbar	proj
rasterized	text	ticklabels	ticks	ticksize
tickweight	tight	title	titlesize	titleweight
windplot				

Table 5.2: List of wind specific formatoption keywords

scale	arrowstyle	density	color	lengthscale
reduceabove	streamplot	arrowsize	enable	linewidth
clabel				

Table 5.3: List of LinePlot specific formatoption keywords

axiscolor	enable	figtitle	figtitlesize	figtitleweight
fontsize	fontweight	grid	labelsize	labelweight
legend	scale	text	ticksize	tickweight
tight	title	titlesize	titleweight	xlabel
xlim	xrotation	xticklabels	xticks	ylabel
ylim	yrotation	yticklabels	yticks	

- If string s: this will be used as (1., 1., s, “ha”: “right”) (i.e. a string in the upper right corner of the axes).
- If tuple or list of tuples, each tuple defines a text instance on the plot. 0≤x, y≤1 are the coordinates. The coord.-system can be either the data coordinates (default, “data”) or the axes coordinates (“axes”) or the figure coordinates (“fig”). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. “color”, “fontweight”, “fontsize”, etc., see matplotlib.text.Text for possible keys). To remove one single text from the plot, set (x,y,”) for the text at position (x,y); to remove all set text=[]. Metadata keys (var, time, level, or netCDF attributes like long_name, units, ...) maybe replaced via %(key)s. If the time information in the reader (i.e. NetCDF file) is stored in relative (e.g. hours since ...) or absolute (day as %Y%m%d.f) units, directives like %Y for year or %m for the month as given by the python datetime package, are also replaced by the specific time information. There are furthermore some special keys which are replaced when you insert ’key’ in your text (e.g. tinfo). Those are tinfo: %B %d, %Y. %H:%M dinfo: %B %d, %Y Those special keys are defined in the nc2map.defaults.texts[“labels”] dictionary.

lsm: Boolean (Default: True). If True, the continents will be plottet.

merilabelpos: List of 4 values (Default: None) that control whether meridians are labelled where they intersect the left, right, top or bottom of the plot. For example labels=[1, 0, 0, 1] will cause meridians to be labelled where they intersect the left and bottom of the plot, but not the right and top.

ticksize: string or float (Default: small). Defines the size of the ticks (see fontsize for possible values)

labelsize: string or float (Default: medium). Defines the size of x- and y-axis labels (see fontsize for possible values)

latlon: True/False (Default: True). Sets latlon keyword for basemap plot function (or not).

figtitleweight: Fontweight of the figure suptitle (Default: Defined by fontweight property). See fontweight above for possible values.

titleweight: Fontweight of the title (Default: Defined by fontweight property). See fontweight above for possible values. This is the title of this specific axes! For the title of the figure see figtitleweight

lonlatbox: 1D-array [lon1,lon2,lat1,lat2], string (or pattern), or dictionary (Default: global, i.e. [-180.0, 180.0, -90.0, 90.0] for proj=="cyl" and Northern Hemisphere for "northpole" and Southern for "southpole"). Selects the region for the plot.

- If string this will be compiled as a pattern to match any of the keys in nc2map.defaults.lonlatboxes (it contains longitude-latitude definitions for countries and continents). E.g. to focus on Germany, set lonlatbox="Germany". To focus on Africa, set lonlatbox="Africa". To focus on Germany, France and Italy, set lonlatbox='Germany—France—Italy'.
- If dictionary possible keys are
  - "ifile" to give an input shapefile (if not set, use the shapes from the default shape file located at /home/chilipp/Dokumente/myplots-scripts/nc2map/data/countries_and This Shapefile is based upon the bnd-political-boundary-a.shp shapes from the Vmap0 Dataset from GIS-Lab (http://gis-lab.info/qa/vmap0-eng.html), accessed May 2015.
  - any field name in the input shape file (see nc2map.get_fnames and nc2map.get_unique_vals function) to select specific shapes

labelweight: Fontweight of axis labels (Default: Defined by fontweight property). See fontweight above for possible values.

mask: array (x[, var][, num]) (Default: None). The first entry must be a string for a netCDF file or a nc2map.readers.ArrayReader instance, the second entry might be the name of the variable in the mask file to read in, the number at the end defines the values where to mask (if not given, mask everywhere where the mask is 0)

tight: Boolean (Default: False). Make tight_layout after plotting if True.

proj: string ("cyl", "robin", "northpole", "southpole") or dictionary (Default: cyl). Defines the options for the projection used for the plot. If string, Basemap is set up automatically with settings from lonlatbox, if dictionary, these are the keyword arguments passed to mpl_toolkits.basemap.Basemap initialization.

titlesize: string or float (Default: large). Defines the size of the title (see fontsize for possible values)

parallels: 1D-array or integer (Default: 5). Defines the lines where to draw parallels. Possible types are

- 1D-array: manually specify the location of the parallels
- integer: Gives the number of parallels between maximal and minimal lattitude (including max- and minimum line)

fontweight: A numeric value in the range 0-1000 or string (Default: None). Defines the fontweight of the ticks. Possible strings are one of “ultralight”, “light”, “normal”, “regular”, “book”, “medium”, “roman”, “semibold”, “demibold”, “demi”, “bold”, “heavy”, ’extra bold’, “black”.

paralabelpos: List of 4 values (Default: None) that control whether parallels are labelled where they intersect the left, right, top or bottom of the plot. For example labels=[1, 0, 0, 1] will cause parallels to be labelled where they intersect the left and and bottom of the plot, but not the right and top.

grid: Enables the plotting of the grid on the axes if not None (Default: False).

axiscolor: string or color for axis or dictionary (Default: “top”: None, “right”: None, “bottom”: None, “left”: None). If string or color this will set the default value for all axis. If dictionary, keys must be in [“right”, “left”, “top”, “bottom”] and the values must be a string or color to set the color for “right”, “left”, “top” or “bottom” specificly.

ocean_color: color instance (Default: None). Specify the color of the ocean. Attention! Might reduce the performance a lot if multiple plots are opened! To not kill everything, use the MapBase.share.lsmask method of the specific MapBase instance.

lineshapes: string, list of strings or dictionary. (Default: None). Draw polygons on the map from a shapefile.

- If string or list of strings this will be seen as the values for the COUNTRY field in the default shapefile (see “ifile” below) and all matching polygons in this shape file will be merged.
- If dictionary possible keys are
  - “ifile” to give an input shapefile (if not set, use the shapes from the default shape file located at /home/chilipp/Dokumente/myplots-scripts/nc2map/data/cou This Shapefile is based upon the bnd-political-boundary-a.shp shapes from the Vmap0 Dataset from GIS-Lab (http://gis-lab.info/qa/vmap0-eng.html), accessed May 2015.

- "ofile" for the target shape file if specific shapes are selected or "dissolve" is set to False
- "dissolve". True/False (Default: False). If True, all polygons will be merged into one single shape
- any field name in the input shape file (see nc2map.get_fnames and nc2map.get_unique_vals function) to select specific shapes
- any other key (but the "name" key) which is finally passed to the read-shapefile method (e.g. "color" or "linewidth")
  Each shape is uniquely defined through a key. If you use a dictionary d with the settings described above, you can set the key manually via 'my_own_key': d. Otherwise a key like 'shape%i' will automatically be assigned, where '%i' depends on the number of already existing shapes. You can use these keys to remove a shape from the current plot by simply setting shapes='key_to_remove' (or whatever key you want to remove). Otherwise you can remove all drawn shapes with anything that evaluates to False (e.g. shapes=None). Please note that it might take a while to dissolve all polygons if "dissolve" is set to True and even to extract them if the shapefile is large. Therefore, if you use the shape on multiple plots, use the share.lineshapes method of the specific MapBase instance

rasterized: Boolean (Default: True). Rasterize the pcolormesh (i.e. the mapplot) or not.

countries: Boolean (Default: False). If True, draw country borders.

title: string (Default: None). Defines the title of the plot. Metadata keys (var, time, level, or netCDF attributes like long_name, units, ...) maybe replaced via %(key)s. If the time information in the reader (i.e. NetCDF file) is stored in relative (e.g. hours since ...) or absolute (day as %Y%m%d.f) units, directives like %Y for year or %m for the month as given by the python datetime package, are also replaced by the specific time information. There are furthermore some special keys which are replaced when you insert 'key' in your text (e.g. tinfo). Those are tinfo: %B %d, %Y. %H:%M dinfo: %B %d, %Y Those special keys are defined in the nc2map.defaults.texts["labels"] dictionary. This is the title of this specific axes! For the title of the figure see figtitle

meridionals: 1D-array or integer (Default: 7). Defines the lines where to draw meridionals. Possible types are

- 1D-array: manually specify the location of the meridionals
- integer: Gives the number of meridionals between maximal and minimal longitude (including max- and minimum line)

land_color: color instance (Default: None). Specify the color of the land.

tickweight: Fontweight of ticks (Default: Defined by fontweight property). See fontweight above for possible values.

figtitle: string (Default: None). Defines the figure suptitle of the plot.

fontsize: string or float (Default: None). Defines the default size of ticks, axis labels and title. Strings might be 'xx-small', 'x-small', “small”, “medium”, “large”, 'x-large', 'xx-large'. Floats define the absolute font size, e.g., 12

5.2 Colorbar and colormap formatoptions

plotcbar: String or list of possible strings (see below). Default: [“b”]. Determines where to plot the colorbar. Possibilities are “b” for at the bottom of the plot, “r” for at the right side of the plot, “sh” for a horizontal colorbar in a separate figure, “sv” for a vertical colorbar in a separate figure. For no colorbar use ”, None, False, [], etc. A string may be a combination of multiple positions (e.g. “bsh” will draw a colorbar at the bottom of the plot and a separate horizontal one).

extend: string (“neither”, “both”, “min” or “max”) (Default: neither). If not “neither”, make pointed end(s) for out-of-range values. These are set for a given colormap using the colormap set_under and set_over methods.

ctickweight: Fontweight of colorbar ticks (Default: Defined by fontweight property). See fontweight above for possible values.

ticklabels: Array (Default: None). Defines the ticklabels of the colorbar

bounds: 1D-array, tuple or string (Default: (“rounded”, 11)). Defines the bounds used for the colormap. Possible types are

- 1D-array: Defines the bounds directly by giving the values
- tuple (string, N): Compute the bounds automatically. N gives the number of increments whereas string can be one of the following strings
  - “rounded”: Rounds min and maxvalue of the data to the next 0.5-value with respect to its exponent with base 10 (i.e. 1.3e-4 will be rounded to 1.5e-4)
  - “roundedsym”: Same as “rounded” but symmetric around zero using the maximum of the data maximum and (absolute value of) data minimum.
  - “minmax”: Uses minimum and maximum of the data (without rounding)
  - “sym”: Same as “minmax” but symmetric around 0 (see “rounded” and “roundedsym”).

- tuple (string, N, percentile): Same as (string, N) but uses the percentiles defined in the 1D-list percentile as maximum. percentile must have length 2 with [minperc, maxperc]
- string: same as tuple with N automatically set to 11.

cticksize: string or float (Default: medium). Defines the size of the colorbar ticks (see fontsize for possible values)

cmap: string or colormap (e.g.matplotlib.colors.LinearSegmentedColormap) (Default: jet). Defines the used colormap. If cmap is a colormap, nothing will happen. Otherwise if cmap is a string, a colorbar will be chosen. Possible strings are

- 'red_white_blue' (e.g. for symmetric precipitation colorbars)
- 'white_red_blue' (e.g. for asymmetric precipitation colorbars)
- 'blue_white_red' (e.g. for symmetric temperature colorbars)
- 'white_blue_red' (e.g. for asymmetric temperature colorbars)
- any other name of a standard colorbar as provided by pyplot (e.g. "jet","Greens","binary", etc.). Use function nc2map.show_colormaps to visualize them.

ticks: 1D-array or integer (Default: None). Define the ticks of the colorbar. In case of an integer i, every i-th value of the default ticks will be used.

5.3 Masking properties

maskgreater: Float (Default: None). All data greater than this value is masked (see also maskgeq)

maskbetween: Tuple or list (Default: None). Pair (min, max) between which the data shall be masked

maskless: Float (Default: None). All data less than this value is masked (see also maskleq)

maskleq: Float (Default: None). All data less or equal than this value is masked (see also maskless)

maskgeq: Float (Default: None). All data greater or equal than this value is masked (see also maskgreater)

5.4 Windplot specific formatoptions

arrowsize: float (Default: 1.0). Defines the size of the arrows

arrowstyle: string (Default: $-|>$). Defines the style of the arrows (See :class:‘ matplotlib.patches.FancyArrowPatch‘)

clabel: string (Default: None). Defines the label of the colorbar (if plotcbar is True). Metadata keys (var, time, level, or netCDF attributes like long_name, units, ...) maybe replaced via %(key)s. If the time information in the reader (i.e. NetCDF file) is stored in relative (e.g. hours since ...) or absolute (day as %Y%m%d.f) units, directives like %Y for year or %m for the month as given by the python datetime package, are also replaced by the specific time information. There are furthermore some special keys which are replaced when you insert 'key' in your text (e.g. tinfo). Those are tinfo: %B %d, %Y. %H:%M dinfo: %B %d, %Y Those special keys are defined in the nc2map.defaults.texts[“labels”] dictionary.

color: string (“absolute”, “u” or “v”), matplotlib color code or 2D-array (Default: k). Defines the color behaviour. Possible types are

- 2D-array (which has to match the shape of of u and v): The values determine the colorcoding according to “cmap”
- “absolute”, “u” or “v”: a color coding 2D-array is computed and make the colorcode corresponding to the absolute flow or u or v.
- single letter (“b”: blue, “g”: green, “r”: red, “c”: cyan, “m”: magenta, “y”: yellow, “k”: black, “w”: white): Color for all arrows
- float between 0 and 1 (defines the greyness): Color for all arrows
- html hex string (e.g. '#eeefff'): Color for all arrows

density: Float or tuple (Default: 1.0). Value scaling the density of the arrows (1 means no density scaling)

- If float, this is the value for longitudes and latitudes.
- If tuple (x, y), x scales the longitudes and y the latitude. Please note that for quiver plots (i.e. streamplot=False) densities ¿ 1 are not possible. Densities of quiver plots are scaled using the weighted mean. Densities of streamplots are scaled using the density keyword of the pyplot.streamplot function. See also reduceabove for quiver plots.

enable: Boolean (Default: True). Allows the plot on the axes

lengthscale: String (Default: lin). If “log” the length of the quiver plot arrows are scaled logarithmically via speed = $\sqrt{\log(u)^2 + \log(v)^2}$. This affects only quiver plots (i.e. streamplot=False).

linewidth: float, string (“absolute”, “u” or “v”) or 2D-array (Default: 0). Defines the linewidth behaviour. Possible types are

- float: give the linewidth explicitly

- 2D-array (which has to match the shape of of u and v): The values determine the linewidth according to the given numbers
- "absolute", "u" or "v": a normalized 2D-array is computed and makes the colorcode corresponding to the absolute flow of u or v. A further scaling can be done via the "scale" key (see above). Higher "scale" corresponds to higher linewidth.

reduceabove: Tuple or list (perc, pctl) with floats. (Default: None). Reduces the resolution to "perc" of the original resolution if in the area defined by "perc" average speed is higher than the pctl-th percentile. "perc" can be a float $0 \leq f \leq 1$ or a tuple (x, y) in this range. If float, this is the value for longitudes and latitudes. If tuple (x, y), x scales the longitudes and y the latitude. This defines the scaling of the density (see also density keyword). pctl can be a float between 0 and 100. This formatoption is for quiver plots (i.e. streamplot=False) only. To reduce the resolution of streamplots, use density keyword.

scale: Float (Default: 1.0). Scales the length of the arrows. Affects only quiver plots (i.e. streamplot=False).

streamplot: Boolean (Default: False). If True, a pyplot.streamplot() will be used instead of a pyplot.quiver()

5.5 LinePlot specific formatoptions

legend: location value or dictionary (Default: None). Draw a legend on the axes. If string or integer, this will be used for the location keyword. If dictionary, the settings of this dictionary will be used. Possible keys for the dictionary are given in the following parameter list: Parameters ———- loc : int or string or pair of floats, default: 0 The location of the legend. Possible codes are: ========== ========== Location String Location Code ========== ========== "best" 0 'upper right' 1 'upper left' 2 'lower left' 3 'lower right' 4 "right" 5 'center left' 6 'center right' 7 'lower center' 8 'upper center' 9 "center" 10 ========== ========== Alternatively can be a 2-tuple giving "x, y" of the lower-left corner of the legend in axes coordinates (in which case "bbox_to_anchor" will be ignored). bbox_to_anchor : :class:'matplotlib.transforms.BboxBase' instance or tuple of floats Specify any arbitrary location for the legend in 'bbox_transform' coordinates (default Axes coordinates). For example, to put the legend's upper right hand corner in the center of the axes the following keywords can be used:: loc='upper right', bbox_to_anchor=(0.5, 0.5) ncol : integer The number of columns that the legend has. Default is 1. prop : None or :class:'matplotlib.font_manager.FontProperties' or dict The font properties of the legend. If None (default), the current :data:'matplotlib.rcParams' will be used. fontsize : int or float or 'xx-small', 'x-small', "small", "medium", "large", 'x-large', 'xx-large' Controls the font size of the legend. If the value is numeric the

size will be the absolute font size in points. String values are relative to the current default font size. This argument is only used if ‘prop‘ is not specified. numpoints : None or int The number of marker points in the legend when creating a legend entry for a line/:class:‘matplotlib.lines.Line2D‘. Default is “None“ which will take the value from the “legend.numpoints“ :data:‘rcParam<matplotlib.rcParams¿‘. scatterpoints : None or int The number of marker points in the legend when creating a legend entry for a scatter plot/ :class:‘matplotlib.collections.PathCollection‘. Default is “None“ which will take the value from the “legend.scatterpoints“ :data:‘rcParam<mat scatteryoffsets : iterable of floats The vertical offset (relative to the font size) for the markers created for a scatter plot legend entry. 0.0 is at the base the legend text, and 1.0 is at the top. To draw all markers at the same height, set to “[0.5]“. Default “[0.375, 0.5, 0.3125]“. markerscale : None or int or float The relative size of legend markers compared with the originally drawn ones. Default is “None“ which will take the value from the “legend.markerscale“ :data:‘rcParam ¡matplotlib.rcParams¿‘. frameon : None or bool Control whether a frame should be drawn around the legend. Default is “None“ which will take the value from the “legend.frameon“ :data:‘rcParam<matplotlib.rcParams¿‘. fancybox : None or bool Control whether round edges should be enabled around the :class:‘ matplotlib.patches.FancyBboxPatch‘ which makes up the legend’s background. Default is “None“ which will take the value from the “legend.fancybox“ :data:‘rcParam<matplotlib.rcParams¿‘. shadow : None or bool Control whether to draw a shadow behind the legend. Default is “None“ which will take the value from the “legend.shadow“ :data:‘rcParam<matplotlib.rcParams¿‘. framealpha : None or float Control the alpha transparency of the legend’s frame. Default is “None“ which will take the value from the “legend.framealpha“ :data:‘rcParam<matplotlib.rcParams mode : ”expand”, None If ‘mode‘ is set to “”expand”“ the legend will be horizontally expanded to fill the axes area (or ‘bbox_to_anchor‘ if defines the legend’s size). bbox_transform : None or :class:‘matplotlib.transforms.Transform‘ The transform for the bounding box (‘bbox_to_anchor‘). For a value of “None“ (default) the Axes’ :data:‘ matplotlib.axes.Axes.transAxes‘ transform will be used. title : str or None The legend’s title. Default is no title (“None“). borderpad : float or None The fractional whitespace inside the legend border. Measured in font-size units. Default is “None“ which will take the value from the “legend.borderpad“ :data:‘rcParam<matplotlib.rcParams¿‘. labelspacing : float or None The vertical space between the legend entries. Measured in font-size units. Default is “None“ which will take the value from the “legend.labelspacing“ :data:‘rcParam<matplotlib.rcParams¿‘. handlelength : float or None The length of the legend handles. Measured in font-size units. Default is “None“ which will take the value from the “legend.handlelength“ :data:‘rcParam<matplotlib.rcParams¿‘. handletextpad : float or None The pad between the legend handle and text. Measured in font-size units. Default is “None“ which will take the value from the “legend.handletextpad“ :data:‘rcParam<matplotlib.rcParams¿‘. borderaxespad : float or None The pad between the axes and legend border. Measured in font-size units. Default is “None“ which will take the value from the “legend.borderaxespad“

:data:‘rcParam<matplotlib.rcParams¿‘. columnspacing : float or None The spacing between columns. Measured in font-size units. Default is “None“ which will take the value from the “legend.columnspacing“ :data:‘rcParam<matplotlib.rcParams¿‘. handler_map : dict or None The custom dictionary mapping instances or types to a legend handler. This ‘handler_map‘ updates the default handler map found at :func:‘matplotlib.legend.Legend.get_legend_handler_map‘. Notes —– Not all kinds of artist are supported by the legend command. See :ref:‘plotting-guide-legend‘ for details. Examples ——— .. plot:: mpl_examples/api/legend_demo.py

xlabel: string (Default: None). Defines the x-axis label

xlim: tuple (Default: None). Specifies the limits of the x-axis

xrotation: float (Default 0). Degrees between 0 and 360 for which the xticklabels shall be rotated

xticklabels: format string, 1D-array or dictionary (Default: None). Defines the y-axis ticklabels.

- If None, the automatically calculated y-ticklabels will be used.
- If format string (e.g. ’%0.0f’ for integers, ’%1.2e’ for scientific or ’%b’ for the month if time is plotted on the axis.
- If 1D-array, those will be used for the yticklabels. (Note: The length should match to the used yticks
- If dictionary, possible keys are “minor” for minor ticks and “major” for major ticks. Values can be in either of the styles described above. Note: To enable minor ticks, you use the xticks formatoption

xticks: integer, 1D-array or dictionary (Default: None). Defines the x-ticks.

- If None, the automatically calculated x-ticks will be used.
- If integer i, every i-th tick of the automatically calculated ticks will be used.
- If 1D-array, those will be used for the xticks.
- If dictionary, possible keys are “minor” for minor ticks and “major” for major ticks. Values can be in any of the styles described above. Another possible key is “pad” to define the vertical difference between minor and major ticks. By default, those are calculated from the ticksize formatoption

ylabel: string (Default: None). Defines the y-axis label

ylim: tuple (Default: None). Specifies the limits of the y-axis

yrotation: float (Default 0). Degrees between 0 and 360 for which the xticklabels shall be rotated

yticklabels: format string, 1D-array or dictionary (Default: None). Defines the y-axis ticklabels.

- If None, the automatically calculated y-ticklabels will be used.
- If format string (e.g. '%0.0f' for integers, '%1.2e' for scientific or '%b' for the month if time is plotted on the axis.
- If 1D-array, those will be used for the yticklabels. (Note: The length should match to the used yticks
- If dictionary, possible keys are “minor” for minor ticks and “major” for major ticks. Values can be in any of the styles described above. Note: To enable minor ticks, you use the yticks formatoption

yticks: integer, 1D-array or dictionary (Default: None). Defines the y-ticks.

- If None, the automatically calculated y-ticks will be used.
- If integer i, every i-th tick of the automatically calculated ticks will be used.
- If 1D-array, those will be used for the yticks.
- If dictionary, possible keys are “minor” for minor ticks and “major” for major ticks. Values can be in any of the styles described above. Another possible key is “pad” to define the vertical difference between minor and major ticks. By default, those are calculated from the ticksize formatoption

5.6 Miscallaneous formatoptions

windplot: WindFmt instance. (Default initialized by {}). Defines the properties of the wind plot. Can be set either directly via a WindFmt instance or with a dictionary containing the formatoptions (see show_fmtkeys(“wind”, “windonly”)

6 Data management

There are basically three levels in the `nc2map` module.

1. The `reader` level (the data in the NetCDF file)
2. The `MapBase` level (the plot with the extracted data from the `reader`)
3. The `Maps` level (all plots together)

6.1 Data Readers

As stated already in subsection 2.3.1, `nc2map` uses the python `netCDF4.Dataset` and `netCDF4.MFDataset` classes to read from NetCDF files. Therefore those classes are incorporated as `nc2map.readers.NCReader` and `nc2map.readers.MFNCReader` classes, which themselves are subclasses of the `nc2map.reader.ReaderBase` class. This is due to the fact that the `netCDF4` classes provide only the basic data accessing methods. Furthermore for future purposes maybe other formats (e.g. GeoTIFF) may be supported. Anyway, for the `nc2map` module an easier access to the data via the `get_data` method is provided. You can specify the desired datashape (2d, 3d or 4d) and, the variable and further dimensions. Furthermore you can perform arithmetics with those readers (e.g. subtraction, division, etc., see section 7.2).

6.2 `MapBase` instances in `nc2map.Maps`

All `MapBase` instances are stored in the `maps` attribute of the specific `Maps` instance. However there is no need for you to manually figure out which of the `MapBase` instances is the one you need. Instead you can use the `get_maps` method of the `Maps` class and specify what you need via the meta attributes of the variables. For example if you want to get the `MapBase` instances corresponding to the variable `t2m`, simply use

```
mymaps = nc2map.Maps('my-netcdf-file.nc')
mapos = mymaps.get_maps(vlst='t2m')
```

A MapBase instance extracts the two dimensional data with its `get_data` method. The data is then stored in the `data` attribute, together with `time` information, latitude and longitude fields, as well as the `level` information (use the **help** function for details). Furthermore you can access the full data via the `reader` attribute of the `MapBase` instance (see next section 6.1). The data is a `nc2map.readers.DataField` instance, a wrapper around a

`numpy.ma.MaskedArray` providing additional informations (like the `dimensions` that correspond to each axes or the dimension data in the `DataField.dims` attribute) and methods (like a weighted `percentile` method, `fldmean`, `fldstd`, etc.). For a `MapBase` instance `mapo`, you can access the data array simply via `mapo.data[:]`[1] In general, the `MapBase` instances do not reduce the size of the two-dimensional field, but mask all entries that are not needed (e.g. if you use a global NetCDF file but show only a part of the globe with the lonlatbox formatoption keyword). The same holds for the density formatoption for `WindPlot` (if streamplot is set to `False`). This will only mask unneeded entries and visualize the mean of the now masked entries, but will not decrease the size of the array. To permanently decrease the array, use other tools like `cdos`.

[1]Note: The `DataField` class probably will implemented as a subclass of the `numpy.ma.MaskedArray`.

7 Evaluation routines

There are three possible evaluation methods that are incorporated in the `nc2map` module. I will explain the main principles for application, however please look into the `nc2map/demo` directory for direct application examples and use the python **help** function.

7.1 Incorporation of Climate Data Operators

`cdos` are implemented via the `nc2map.Cdo` class, which itself is based upon the `Cdo` class of the `cdo.py` python module[1]. There are four new keywords implemented for each operator:

returnMaps takes None, a string or list of strings (the variable names) or a dictionary (keys and values are determined by the `Maps` method). None will open maps for all variables that have longitude and latitude dimensions in it. An open `Maps` instance is returned.

returnMap takes a string (the variable name) or a dictionary (keys and values are determined by the `FieldPlot` method). An open `FieldPlot` instance is returned

returnLine takes a string or list of strings (the variable names) or a dictionary (keys and values are determined by the `LinePlot` method). An open `LinePlot` instance is returned

returnData takes a string or list of strings (the variable names) or a dictionary (keys and values are determined by the `get_data` method). It returns a `DataField` instance of the specified variables, with datashape='any'.

Hence you can not only evaluate your results with cdos, but also immediately visualize the data. See the `nc2map/demo/cdo_demo.py` for a demonstration of the possibilities.

7.2 Calculation

As stated in chapter 6, there exist basically three levels. On the lowest two levels (the `MapBase` and `reader` level), you can perform calculations like multiplication, subtraction, power, division or addition. The great advantage is, that you immediately can visualize your result, change meta data, etc. You can find a demo file in `nc2map/demo/calculation_demo.py`.

There are however a few rules that you should consider concerning arithmetics between `MapBase` instances:

[1] `https://code.zmaw.de/projects/cdo/wiki/Cdo{rbpy}`

1. When you apply arithmetics with MapBase instances, the MapBase first extracts it's data in it's `reader` and creates a new reader where it now takes the data from. Therefore, the new reader will (for example) have only one single time step, one single level, etc..

2. You can add floats, `numpy.ndarrays` matching the shape of the `MapBase.data` attribute, other `MapBase` instances[2] or other `readers`.

3. The resulting `MapBase` will have set all dimensions set to 0 (this does only matter, if you perform arithmetics with a `reader`).

If you want to perform arithmetics between readers (e.g. to consider the full data and not only one two-dimensional array), there are also some rules:

1. You can add floats, `numpy.ndarrays` matching the shape of the `MapBase.data` attribute or other `readers`. If you have a `numpy.ndarrays`, the shape has to match the shape of the variables in the reader (this implies that all variables that are not dimensional data (e.g. `time`) must have the same shape).

2. If you add another `reader`, it can have
   a) all the same variables or only one variable which will then be added to the variables of the other `reader`
   b) only one time step which will then be added to all the other time steps in the other `reader`
   c) only one level which will then be added to all the other levels in the other `reader`

It has not been evaluated so far with large data sets, but feel free to do it and I would be happy for results :) However it is generally faster to make calculations on the `reader` level than on the `MapBase` level.

7.3 Evaluator classes

Additionally to the cdo interface (see section 7.1) there are (currently) two evaluators implemented (the `FldMeanEvaluator` and `ViolinEvaluator`), which you can access via the `evaluate` method of your `Maps` instance. Those evaluators both possibly can evaluate multiple regions at the same time. The region definition is thereby determined by the mask formatoption keyword. However you can of course simply zoom to the region that you are interested in (see the lonlatbox keyword) and make an evaluation without specifying the regions.

For an example look into the `nc2map/demo/evaluators_demo.py` script.

[2] you can only calculate between scalar fields (i.e. `FieldPlot` with `FieldPlot`) or between vector fields (i.e. `WindPlot` with `WindPlot`), but not mix the two classes

Glossary

C | D | E | F | G | L | M | N | R | S | V

C

CDOs (`nc2map.Cdo`) Climate Data Operators. They can be optionally used in combination with the NetCDF package. See `https://code.zmaw.de/projects/cdo` for a documentation. 26, 27

D

DataField (`nc2map.readers.DataField`) Basic data class in the `nc2map` module containing the variable data as well as dimension informations (latitude, longitude, time, level, etc.). 25–27, 30, *see* `nc2map.reader.ReaderBase.get_data`

Dimension identifiers ii, 5, 30, 31

- **level** Identifier for the vertical level (as integer). 3, 5, 7, 9, 25
- **name** Identifier for the name of the `MapBase` instance. 3, 5, 7, 9
- **time** Identifier for the time (as integer). 3, 5, 7, 9, 25, 28
- **var** (plural: `vlst`). Identifier for the variable. 3, 7, 9

E

evaluator (`nc2map.evaluators.EvaluatorBase`) Generally a subclass of the `EvaluatorBase` class, which is designed to evaluate multiple `MapBase` instances.. *see* `nc2map.Maps.evaluate`

- **FldMeanEvaluator** (`nc2map.evaluators.FldMeanEvaluator`) Calculates the weighted 2-dimensional field mean (i.e. the mean over all longitudes and latitudes), saves the data into new `ArrayReader` instances and creates `LinePlot` instances that show the data. You can do that for multiple regions at the same time and include errors. The key in the `evaluate` method is `'fldmean'`. 28, 31
- **ViolinEvaluator** (`nc2map.evaluators.ViolinEvaluator`) Creates `ViolinPlot` instances that show the data. You can do that for multiple regions at the same time. The key in the `evaluate` method is `'violin'`. 28, 31, 32, *see* `nc2map.mapos.ViolinPlot`

F

formatoption (`fmt`) Formatoption keywords that control the appearance of the plot (see chapter 5 for details).. ii, 4, 6, 8, 13, 28, 30

- **get_fmtdocs** (`nc2map.get_fmtdocs`) Same as `show_fmtdocs`, but instead of displaying the documentation, returns a dictionary with formatoption keywords as keys and their documentation as value.. 8, 30, *see* `nc2map.show_fmtdocs`

get_fmtkeys (`nc2map.get_fmtkeys`) Same as `show_fmtkeys`, but returns a string.. 8, 30, *see* `nc2map.show_fmtkeys`

show_fmtdocs (`nc2map.show_fmtdocs`) Helper function to display the possible formatoption keywords and their documentation. 8, 13, 29, *see* `nc2map.get_fmtdocs`

show_fmtkeys (`nc2map.show_fmtkeys`) Helper function to display the possible formatoption keywords. 8, 13, 30, *see* `nc2map.get_fmtkeys`

G

get_fnames Displays all possible field names that are in the default shape file used by the lineshapes formatoption.. 8, 30, *see*

get_unique_vals Displays all possible values that are in the default shape file used by the lineshapes formatoption.. 8, 30, *see*

L

LinePlot (`nc2map.mapos.LinePlot`) Class to visualize one dimensional data in the NetCDF file. 8, 13, 27, 29–32

M

MapBase (`nc2map.mapos.MapBase`) Basic class that reads data from a NetCDF file and plots visualizes it on one axes. 2, 3, 5–10, 12, 25–32

FieldPlot (`nc2map.mapos.FieldPlot`) Class that plots a two-dimensional field (e.g. temperature) and optionally overlained by a wind field. 2, 13, 27, 28

data (`FieldPlot.data`) `DataField` instance storing the data of the variable. 25

MapBase.get_data (`nc2map.mapos.MapBase.get_data`) Method of `MapBase` (and `LinePlot`) instances to extract the data from the reader. 25, *see* `nc2map.reader.ReaderBase.get_data`

meta (`MapBase.meta`) meta data property of the `MapBase` instance which gives a dictionary containing all the meta data information of the specific variable. 9, 30

WindPlot (`nc2map.mapos.WindPlot`) Class that plots a wind field (stream plot or quiver plot). 2, 13, 26, 28

MapsManager (`nc2map.MapsManager`) Basic class that controls multiple `MapBase` instances. 2, 30, 31

addline (`MapsManager.addline`) Add a new LinePlot instance to the `MapsManager` instance. This function is used at the initialization of a `Maps` instance if `linesonly=True` is set.. 6

addmap (`MapsManager.addmap`) Add a new MapBase instance to the `MapsManager` instance. This function is used at the initialization of a `Maps` instance.. 6

get_label_dict (`Maps.get_label_dict`) Helper function. Returns the meta data of the given `MapBase` instance. 9, *see* & `MapBase.meta`

get_maps (`MapsManager.get_maps`) Helper function. Returns the `MapBase` corresponding to the given dimension identifier. 25

Maps (`nc2map.Maps`) Basic class for plotting in `nc2map`. Controls multiple `MapBase` instances. 2, 4–9, 11, 13, 25, 27, 28, 30, 31

evaluate (`nc2map.Maps.evaluate`) Evaluator method that passes `MapBase` instances to evaluators. 28, 29, *see* `nc2map.evaluators.FldMeanEvaluator` & `nc2map.evaluators.ViolinEvaluator`

make_movie (`Maps.make_movie`) Movie method of the `Maps` class. Exports the chosen `MapBase` instances (or figures) into a movie of the specified format for the user given (or all) time steps. `MapBase` instances may be chosen via dimension identifiers. 8, 31, *see* `Maps.output`

nextt (`Maps.nextt`) Updates all MapBase instances controlled by the `Maps` instances to the next time step. 6, 31, *see* `Maps.prevt`

output (`Maps.output`) Output method of the `Maps` class. Exports the chosen `MapBase` instances (or figures) into different (static) formats. `MapBase` instances may be chosen via dimension identifiers. 8, 31, *see* `Maps.make_movie`

prevt (`Maps.prevt`) Updates all MapBase instances controlled by the `Maps` instances to the next time step. 6, 31, *see* `Maps.nextt`

save (`Maps.save`) Helper function of the `Maps` class. This method creates a `pickle` file that can be loaded with the `load` function, to reinitialize of `Maps` instance with all coordinated `MapBase` instances and their settings. 8, 31, *see* `nc2map.load`

update (`Maps.update`) Update method of the `Maps` class. Updates the chosen `MapBase` instances by the given formatoptions. `MapBase` instances may be chosen via dimension identifiers. 3, 4, 6, 12, 13

meta Property that returns a dictionary containing all the meta information in the `MapBase` and `LinePlot` instances of the `MapsManager` instance. 9, 30

N

nc2map Interactive python module to visualize NetCDF files on a map. 2, 4, 8, 9, 11–13, 25, 27, 29–32

load (`nc2map.load`) Function that loads `pickle` files generated by the `save` method and opens the `Maps` instance with the previously saved settings. 8, 31, *see* `Maps.save`

R

reader (`nc2map.reader.ReaderBase`) General identifier for a `nc2map.readers.ReaderBase` instance. 4, 11, 25, 27, 28, 31, 32

ArrayReader (`nc2map.readers.ArrayReader`) Base class for the data management in `nc2map`. This class is initialized by the raw data stored in a dictionary. 4, 29

get_data (`nc2map.reader.ReaderBase.get_data`) `get_data` method of the `reader` class to easiliy access the data stored in the `reader` (e.g. `NCReader` or `MFNCReader`) instance. 25, 27, 29, 30

MFNCReader (`nc2map.readers.MFNCReader`) `reader` subclass based upon the `netCDF4.MFDataset` class to read multiple NetCDF files. 4, 25, 31

NCReader (`nc2map.readers.NCReader`) `reader` subclass based upon the `netCDF4.Dataset` class to read one single NetCDF file. 4, 25, 31

S

show_colormaps (`nc2map.show_colormaps`) Displays all predefined colormaps (or the one specified by the user). 8

shp_utils (`nc2map.shp_utils`) Shape file module of the `nc2map` module. 8

V

ViolinPlot (`nc2map.mapos.ViolinPlot`) Class to make a violin plot using the python `seaborn.violinplot` function. Note that if you want to create an `ViolinPlot` instance manually, this class (different from `LinePlot` and `MapBase` instances) does not extract the data from a `reader`. Instead you have to pass it in manually at the initialization. 29, *see* `nc2map.evaluators.ViolinEvaluator`