Gplot

Release v0.3a

Guangzhi XU

Dec 07, 2022

CONTENTS

Table of Contents

CHAPTER

ONE

INTRODUCTION

Gplot is a (thin) wrapper around *matplotlib*, *basemap* and *cartopy* for quick and easy creations of geographical plots. It is designed to create publish-ready figures with as few lines as possible, while preserving the possibility to fine-tune various aspects of the plots.

CHAPTER

TWO

INSTALLATION

2.1 Install from conda

`gplot` can be installed in an existing conda environment using:

```
conda install -c guangzhi gplot
```

This will install `gplot` and its dependencies for Python 3.

CHAPTER

THREE

DEPENDENCIES

- Mandatory:
  - OS: Linux or MacOS. Windows is not tested.
  - *Python*: >= 3.
  - *numpy*
  - *matplotlib*: developed in 3.2.2. **NOTE** that versions later than 3.2.2 are incompatible with *basemap*.
- Optional:
  - *scipy*: optional, developed in 1.2.1. For 2D interpolation in quiver plots only.
  - For plotting the geography: *basemap* or *Cartopy*.
    - *basemap*: developed in 1.2.0.
    - *Cartopy*: developed in 0.16.0, not fully supported yet.
  - For *netCDF* file reading: *netCDF4* or *CDAT* or *xarray* or iris.
    - *netCDF4*: developed in 1.5.5.1.
    - the *cdms* module of *CDAT*: developed in 3.1.5.
    - *xarray*: not supported yet.
    - *iris*: not supported yet.

CHAPTER

FOUR

QUICK START

After installation of *gplot* and *basemap*, create an isofill/contourf plot of the global sea level pressure field (sample data included in the installation) using the following snippet:

```
import matplotlib.pyplot as plt
import gplot
from gplot.lib import netcdf4_utils

var = netcdf4_utils.readData('msl')
lats = netcdf4_utils.readData('latitude')
lons = netcdf4_utils.readData('longitude')

figure = plt.figure(figsize=(12, 10), dpi=100)
ax = figure.add_subplot(111)
iso = gplot.Isofill(var)
gplot.plot2(var, iso, ax, xarray=lons, yarray=lats,
            title='Default basemap', projection='cyl',
            nc_interface='netcdf4')
figure.show()
```

The output is given below:

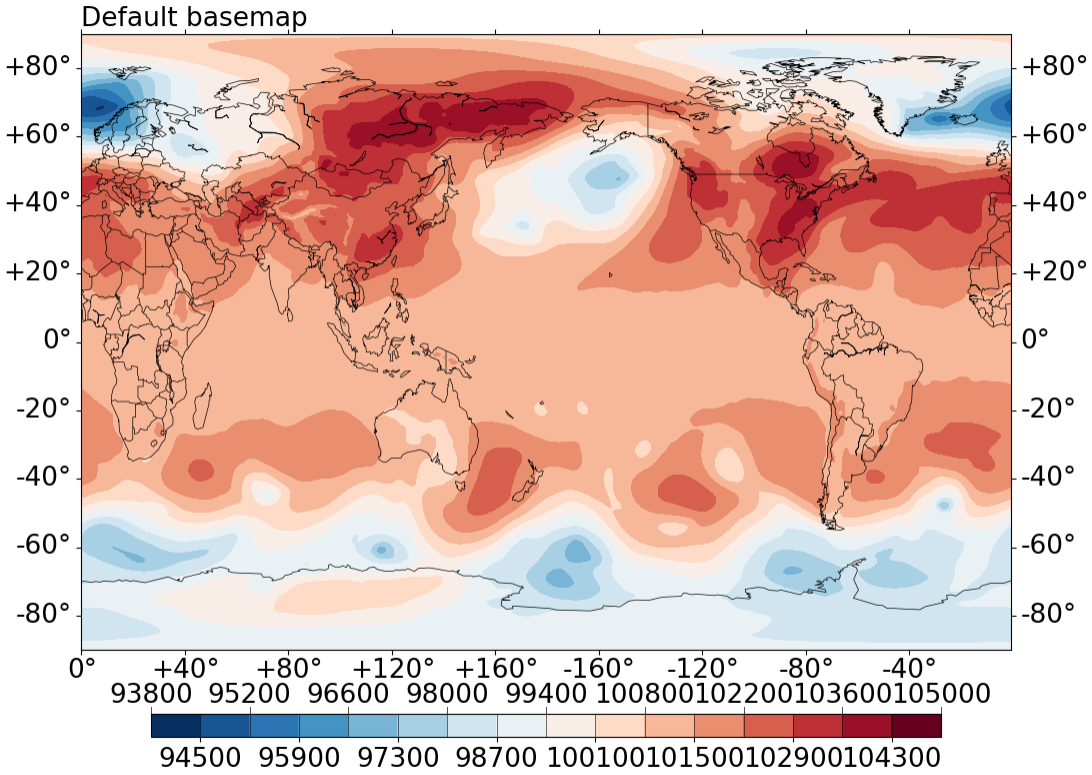

Fig. 4.1: Default contourf plot of global surface pressure field (in Pa), from ERA-I.

CHAPTER

FIVE

DOCUMENTATION

5.1 Basic plots

Table of Contents

5.1.1 Overall design of gplot

The overarching structure of *gplot* is pretty simple (see Fig.5.1): there are 2 major plotting classes, *Plot2D* and *Plot2Quiver*. The latter is specifically for 2D quiver plots, and the former handles commonly used 2D visualization types, including

- isoline/contour
- isofill/contourf
- boxfill/imshow/pcolormesh
- hatching

These 2 classes accept *ndarray* as inputs, which can be provided by 4 widely used *netCDF* file I/O modules: *netcdf4*, *CDAT*, *Iris* and *xarray*. Note that these are optional dependencies, and both *Plot2D* and *Plot2Quiver* work for plain *ndarray* data as well.

Note: Only *netcdf4* and *CDAT* are currently supported. For the latter, only its *cdms2* module is required.

On top of *Plot2D* and *Plot2Quiver*, plotting with geographical map projections are supported by utilizing *basemap* or *Cartopy*, giving rise to 4 derived classes:

- **Plot2Basemap**: 2D plots as *Plot2D* but using *basemap* as the "backend" for geographical map projections.
- **Plot2QuiverBasemap**: 2D quiver plots as *Plot2Quiver* but using *basemap* as the "backend" for geographical map projections.
- **Plot2Cartopy**: 2D plots as *Plot2D* but using *Cartopy* as the "backend" for geographical map projections.
- **Plot2QuiverCartopy**: 2D quiver plots as *Plot2Quiver* but using *Cartopy* as the "backend" for geographical map projections.

Note: *basemap* has been deprecated, however, *Cartopy* is not fully mature in terms of features and robustness. In *gplot*, more attention is paid on *basemap* plots, and the *Cartopy* counterparts are largely work-in-process at the moment.

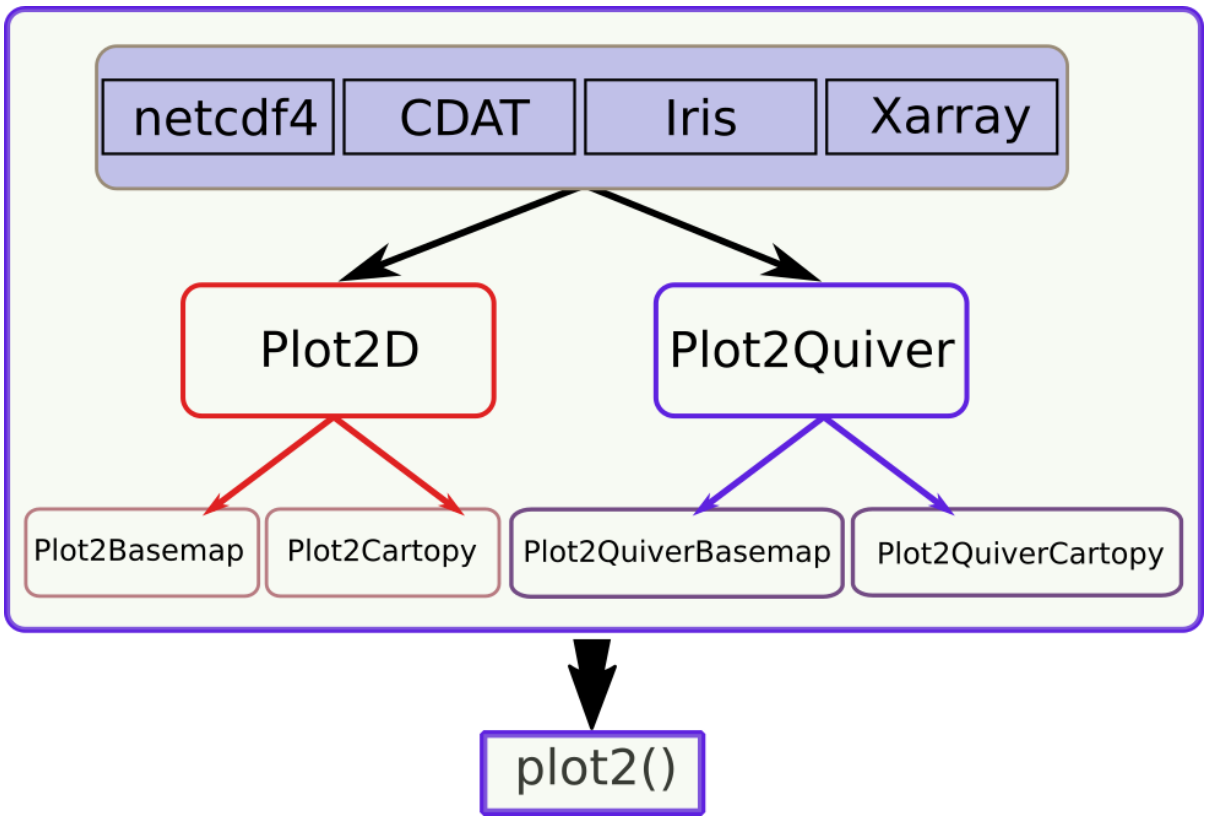

Fig. 5.1: Overarching structure of gplot.

5.1.2 Basic plotting syntax

To give an example of using `Plot2Basemap`:

```
figure = plt.figure(figsize=(12, 10))
ax = figure.add_subplot(111)
iso = gplot.Isofill(var)
gp = Plot2Basemap(var, iso, lons, lats, ax=ax)
gp.plot()
figure.show()
```

where:

- `var` is the `ndarray` input data to be plotted.
- `iso` is an `Isofill` object, which defines an isofill/contourf plot. More details on `Isofill` are given in *Create isofill/contourf plots*.
- `lons` and `lats` give the longitude and latitude coordinates, respectively. They define the geographical domain to generate the map.

This also illustrates the basic "syntax" of *gplot*'s plotting function: there are 3 major elements that define a plot:

1. `var`: the input array – what to plot,
2. `iso`: the plotting method – how to plot, and
3. `ax`: the *matplotlib axis* object – where to plot.

Similarly, for a 2D quiver plot example:

```
figure = plt.figure(figsize=(12, 10))
ax = figure.add_subplot(111)
q = gplot.Quiver(step=8)
pquiver = Plot2QuiverBasemap(u, v, q, xarray=lons, yarray=lats,
    ax=ax, projection='cyl')
pquiver.plot()

figure.show()
```

Note that in this case, there are 2 input arrays (`u` and `v`), the u- and v- velocity components. And `q = gplot.Quiver(step=8)` defines the plotting method.

With these 3 basic elements – input array, plotting method and axis – provided, *gplot* will try to handle the remaining trifles for you, including the axes ticks and labels, colorbar, subplot numbering etc..

Lastly, there is also a `plot2()` interface function in `gplot` that wraps everything in a single function call. To reproduce the 1st example above, one can use:

```
figure = plt.figure(figsize=(12, 10))
ax = figure.add_subplot(111)
iso = gplot.Isofill(var)
gplot.plot2(var, iso, ax, xarray=lons, yarray=lats)
figure.show()
```

And the 2nd example can be achieved using:

```
figure = plt.figure(figsize=(12, 10))
ax = figure.add_subplot(111)
q = gplot.Quiver(step=8)
gplot.plot2(u, q, ax, xarray=lons, yarray=lats, var_v=v,
    projection='cyl')
figure.show()
```

Note that the v- component has been provided using the `var_v` keyword argument.

These design choices are taken to achieve the primary goal of *gplot*, which is to help create good enough plots as quickly and easily as possible.

5.2 Create isofill/contourf plots

Table of Contents

5.2.1 The `Isofill` class

To create an isofill/contourf plot, one creates a *base_utils.Isofill* object as the plotting method, and passes it to the *base_utils.Plot2D* constructor or the *base_utils.plot2()* function.

5.2.2 Define the contour levels

One key element of a good isofill/contourf plot is a set of appropriately chosen contour levels. There are basically 2 ways to define the contour levels in *base_utils.Isofill*:

1. Automatically derive from input data, and a given number of levels

Your data may come with various orders of magnitudes, and sometimes it can be a bit tricky (and annoying) to manually craft the contour levels for each and every plot you create, particularly when you just want to have a quick read of the data. The 1st approach comes as handy for such cases.

To automatically derive the contour levels, these input arguments to the constructor of *base_utils.Isofill* are relevant:

- `vars`: input data array(s).

  The 1st and only mandatory input argument is `vars`, which is the input `ndarray`, or a list of arrays to be plotted. This is used to determine the value range of the input data. Missing values (masked or `nan`) are omitted.

  The list input form is useful when one wants to use the same set of contour levels to plot multiple pieces of data.

- `num`: the **desired** number of contour levels.

  Note that in order to derive nice-looking numbers in the contour levels, the resultant number may be sightly different.

  What is meant by "nice-looking" is that the contour level values won't be some floating point numbers with 5+ decimal places, like what one would get using, for instance

```
>>> np.linspace(0, 30, 12)
array([ 0.        ,  2.72727273,  5.45454545,  8.18181818, 10.90909091,
       13.63636364, 16.36363636, 19.09090909, 21.81818182, 24.54545455,
       27.27272727, 30.        ])
```

  Instead, *base_utils.Isofill* would suggest something like this:

```
[0.0, 2.5, 5.0, 7.5, 10.0, 12.5, 15.0, 17.5, 20.0, 22.5, 25.0, 27.5, 30.0]
```

- `zero`: whether `0` is allowed to be one contour level.

  `zero = 0` exerts no inference on the inclusion of `0`.

`zero = -1` prevents the number `0` from being included in the contour levels, instead, there would be a 0-crossing contour interval, e.g. `[-2, 2]`, that represent the 0-level with a range.

This is very helpful in plots with a divergent colormap, e.g. `plt.cm.RdBu`. Your plot will have a white contour interval, rather than just various shades of blues and reds. The white area represents a kind of buffer zone in which the difference is not far from 0, and the plot will almost always end up being cleaner.

- `min_level`, `max_level`, `ql`, `qr`: determine the lower and upper bounds of the data range to plot.

  `min_level` and `max_level` are used to specify the *absolute* bounds. If `None` (the default), these are taken from the minimum and maximum values from `vars`.

  `ql` and `qr` are used to specify by **relative** bounds: `ql` for the left quantile and `qr` for the right quantile. E.g. `ql = 0.01` takes the `0.01` left quantile as the lower bound, and `qr = 0.05` takes the 0.95 quantile as the upper bound. These are useful for preventing some outliers from inflating the colorbar.

  If both `ql` and `min_level` are given, whichever gives a greater absolute value is chosen as the lower bound. Similarly for `qr` and `max_level`.

Note: In order to arrive at nice-looking contour level numbers, the resultant bounds may not be exactly as requested.

If the lower/upper bound does not cover the entire data range, an **extension** on the relevant side is activated:

```
self.ext_1 = True if self.data_min < vmin else False
self.ext_2 = True if self.data_max > vmax else False
```

These will be visually represented as an **overflow** on the colorbar.

2. Manually specify the contour levels.

Manual contour levels are simply specified by the *levels* keyword argument:

```
iso = Isofill(var, 10, levels=np.arange(-10, 12, 2))
```

This will override the effects from all the arguments listed in the above section, except that overflows will still be added, if your specified levels do not cover the entire data range.

5.2.3 Choose the colormap

The colormap is specified using the `cmap` argument, which is default to a blue-white-red divergent colormap `plt.cm.RdBu_r`.

To use a different colormap, provide one from the *matplotlib*'s colormap collection, e.g. `cmap = plt.cm.rainbow`. It is possible to give only the name of the colormap as a string: `cmap = 'rainbow'`.

5.2.4 Split the colormap colors

Divergent colormaps are commonly used in academic works. The `plt.cm.RdBu_r` colormap is one such example, with a transition from dark blue (the minimum) to white in the middle, and to dark red (the maximum) on the right.

The middle color (white in this case) usually corresponds to some critical transition in the data (e.g. going from negative to positive), therefore it is crucial to make sure they are aligned up. See an example:

```
import matplotlib.pyplot as plt
import gplot
from gplot.lib import netcdf4_utils

# read in SST data
var2 = netcdf4_utils.readData('sst')
lats = netcdf4_utils.readData('latitude')
lons = netcdf4_utils.readData('longitude')

var2ano=var2-280.  # create some negative values

figure, axes = plt.subplots(figsize=(12, 10), nrows=2, ncols=2,
        constrained_layout=True)

iso1=gplot.Isofill(var2ano, num=11, zero=1, split=0)
gplot.plot2(var2ano, iso1, axes.flat[0], legend='local',
        title='negatives and positives, split=0')

iso2=gplot.Isofill(var2ano, num=11, zero=1, split=1)
gplot.plot2(var2ano, iso2, axes.flat[1], legend='local',
        title='negatives and positives, split=1')

iso3=gplot.Isofill(var2ano, num=11, zero=1, split=2)
gplot.plot2(var2ano, iso3, axes.flat[2], legend='local',
        title='negatives and positives, split=2')

iso4=gplot.Isofill(var2, num=11, zero=1, split=2)
gplot.plot2(var2, iso4, axes.flat[3], legend='local',
        title='all positive, split=2')

figure.show()
figure.tight_layout()
```

The output is given in Fig.5.2 below:

To summarize:

- `split=0`: do not split the colormap.
- `split=1`: split the colormap if data have both positive and negative values. Do not split if data have only negative or only positive values.
- `split=2`: force split. If the data have both positive and negative values, the effect is the same as `split=1`. If data have only positive (negative) values, will only use the right (left) half of the colormap.

Note: Positive v.s. negative is one way of splitting the data range into 2 halves, at the dividing value of `0`. It is possible to use an arbitray dividing value, by using the `vcenter` argument. E.g. `iso = gplot.Isofill(var, num=10,`

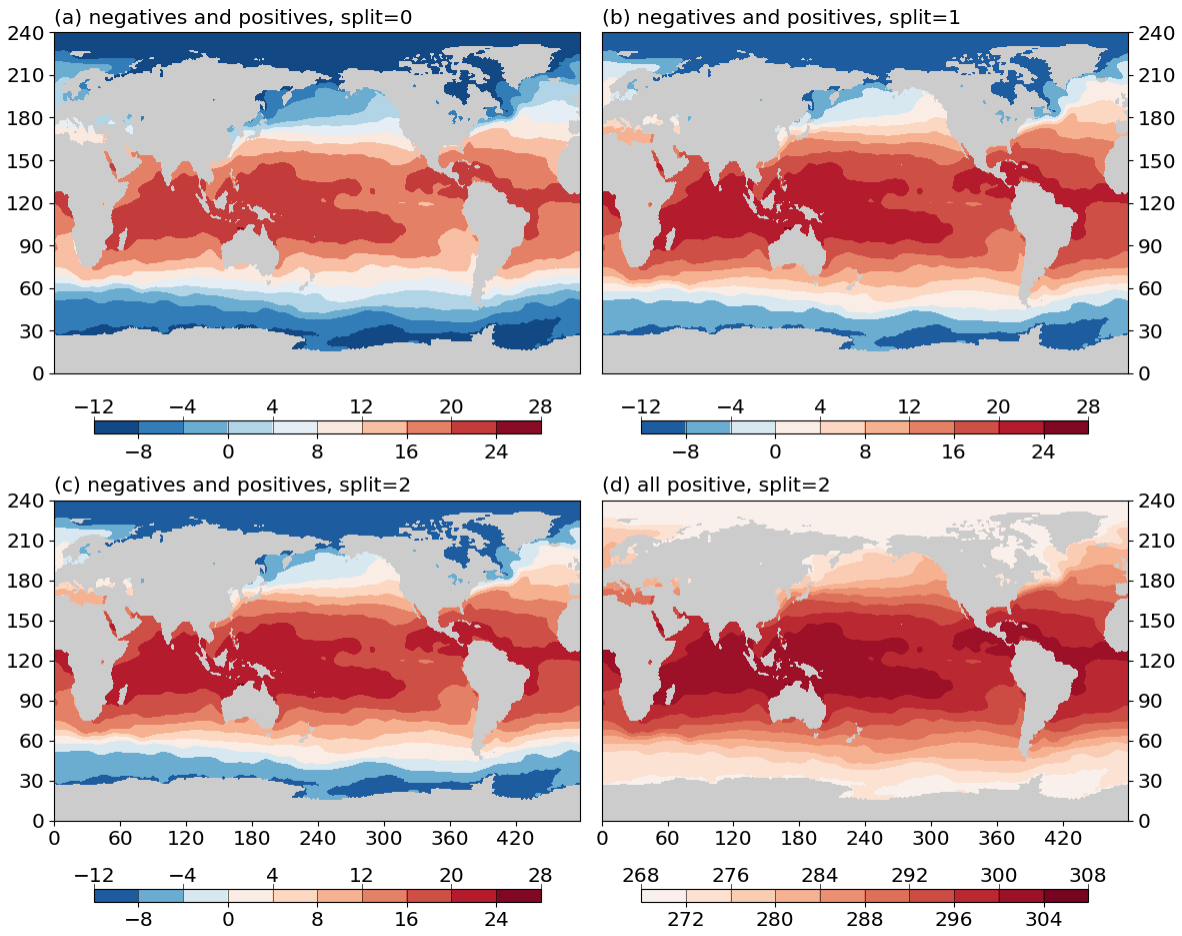

Fig. 5.2: Effects of the `split` argument. (a) do not split the colormap for data with negative and positive values (`split=0`). (b) split the colormap if data have both negative and positive values (`split=1`). (c) force split the colormap when data have both negative and positive values (`split=2`). (c) force split the colormap when data have only positive values (`split=2`).

```
split=2, vcenter=10)
```

5.2.5 Overlay with stroke

It is possible to stroke the isofill/contourf levels with a layer of thin contour lines. E.g.

```
import matplotlib.pyplot as plt
import gplot
from gplot.lib import netcdf4_utils

# read in SLP data
var1 = netcdf4_utils.readData('msl')
lats = netcdf4_utils.readData('latitude')
lons = netcdf4_utils.readData('longitude')

figure, (ax1, ax2) = plt.subplots(figsize=(12, 5), nrows=1, ncols=2,
        constrained_layout=True)

iso1 = gplot.Isofill(var1)
gplot.plot2(var1, iso1, ax1, title='Basemap isofill without stroke',
            projection='cyl')

iso2 = gplot.Isofill(var1, stroke=True)
gplot.plot2(var1, iso2, ax2, title='Basemap isofill with stroke',
            projection='cyl')
figure.show()
```

The result is given in Fig.5.3 below:

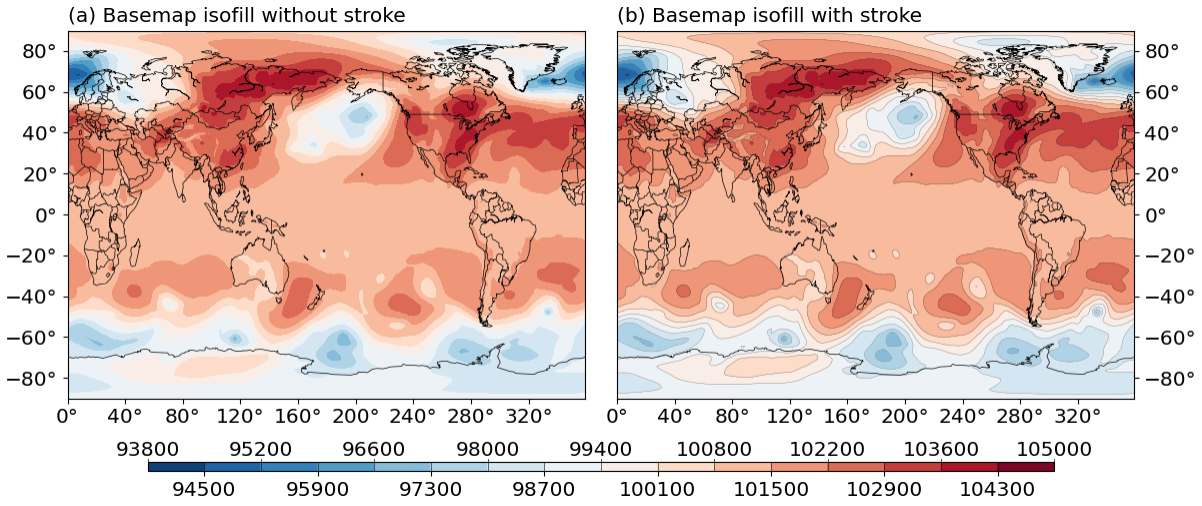

Fig. 5.3: Effects of the `stroke` argument. (a) isofill plot without stroke. (b) isofill plot with stroke.

`stroke` is set to `False` by default. To further control the line width of the stroke, use the `stroke_lw` argument, which is default to `0.2`. The line color is default to a grey color (`stroke_color = 0.3`), and line style default to solid (`stroke_linestyle = '-'`).

5.2.6 The mappable object

gplot calls *matplotlib*'s (or *basemap*'s, if it is using *Plot2Basemap*) `contourf()` function under the hood. The function returns a *mappable object*, e.g. `cs = plt.contourf(data)`. This mappable object is stored as an attribute of the *base_utils.Plot2D* (or *basemap_utils.Plot2Basemap*) object:

```
>>> pobj = Plot2Basemap(var, iso, lons, lats, ax=ax)
>>> pobj.plot()
>>> pobj.cs
<matplotlib.contour.QuadContourSet object at 0x7f0e3e6b4550>
```

The same `plotobj` is returned by the *base_utils.plot2()* function, therefore, the mappable object can be retrieved using:

```
>>> pobj = gplot.plot2(var, iso, ax, xarray=lons, yarray=lats)
>>> pobj.cs
<matplotlib.contour.QuadContourSet object at 0x7f0e3e6b4550>
```

5.3 Create isoline/contour plots

Table of Contents

5.3.1 The `Isoline` class

To create an isoline/contour plot, one creates a *base_utils.Isoline* object as the plotting method, and passes it to the *base_utils.Plot2D* constructor or the *base_utils.plot2()* function.

In many aspects, the *base_utils.Isoline* class is similar as *base_utils.Isofill* (it is in fact derived from the latter). They share these arguments in their `__init__()` methods:

- `vars`
- `num`
- `zero`
- `split`
- `levels`
- `min_level`
- `max_level`
- `ql`

- `qr`
- `vcenter`
- `cmap`

More explanations of these arguments are given in *Create isofill/contourf plots*.

There are a few arguments unique to `Isoline`, and are introduced below.

5.3.2 Line width and color controls

Line width is controlled by the `line_width` input argument, which is default to `1.0`. See Fig.5.4b for an example of changing the line width to a larger value.

Line color, by default, is determined by the colormap (`cmap`). Alternatively, one can use only the black color by specifying `black = True`. Or, use a different color for all contour lines `color = 'blue'`. For single colored isoline plots, the colorbar will not be plotted. See Fig.5.4b,c,d for examples of monochromatic isoline plots.

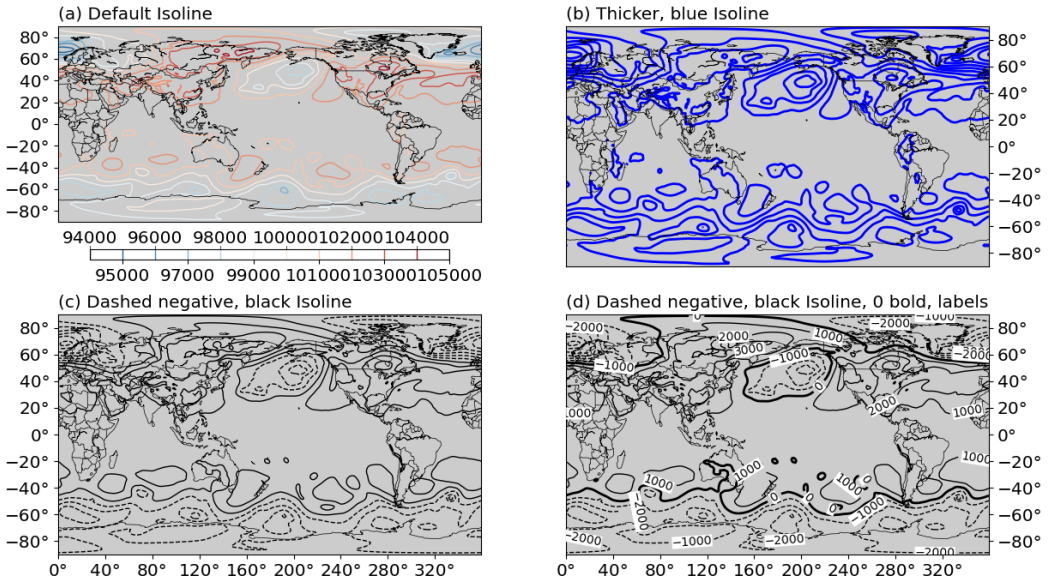

Fig. 5.4: Isoline plot examples. Complete script can be found in `tests.basemap_tests.test_basemap_isolines()` (a) default isoline plot: colored contours, `linewidth=1`. (b) isoline plot with `linewidth=2.0, color='b'`. (c) isoline plot with `black=True, dash_negative=True`. (d) isoline plot with `black=True, dash_negative=True, bold_lines=[0,], label=True, label_box=True`.

5.3.3 Use dashed line for negatives

It is also common to use dashed lines for negative contours and solid lines for positive ones, with optionally a 0-level contour as bold. These can be achieved using:

```
isoline = gplot.Isoline(var, 10, zero=1, black=True, dash_negative=True,
                        bold_lines=[0,])
```

See Fig.5.4c,d for examples.

Note: It is possible to set multiple levels as bold, by specifying them in a list to `bold_lines`.

5.3.4 Label the contour lines

For plots with monochromatic contour lines, one needs to provide a different mechanism for the reading of contour levels, such as labelling out the contours. This can be achieved by passing in the `label = True` argument.

The format of the labels can be controlled by `label_fmt`. If left as `label_fmt = None`, it will use a default `Formatter`. An optional bounding box can be added by `label_box = True`, and one can change the box background color by altering `label_box_color`. See Fig.5.4d for an example.

5.3.5 The mappable object

gplot calls *matplotlib*'s (or *basemap*'s, if it is using *Plot2Basemap*) `contour()` function under the hood. The function returns a *mappable object*, e.g. `cs = plt.contour(data)`. This mappable object is stored as an attribute of the *base_utils.Plot2D* (or *basemap_utils.Plot2Basemap*) object:

```
>>> plotobj = Plot2Basemap(var, iso, lons, lats, ax=ax)
>>> plotobj.plot()
>>> plotobj.cs
<matplotlib.contour.QuadContourSet object at 0x7f0e3e6b4550>
```

The same `plotobj` is returned by the *base_utils.plot2()* function, therefore, the mappable object can be retrieved using:

```
>>> pobj = gplot.plot2(var, iso, ax, xarray=lons, yarray=lats)
>>> pobj.cs
<matplotlib.contour.QuadContourSet object at 0x7f0e3e6b4550>
```

5.4 Create boxfill/imshow plots

Table of Contents

5.4.1 The Boxfill and Pcolor classes

A boxfill/imshow plot is created by defining a *base_utils.Boxfill* plotting method, and passing it to the *base_utils.Plot2D* constructor or the *base_utils.plot2()* function.

All the input arguments to the `__init__()` method of *base_utils.Boxfill* are the same as those in *base_utils.Isofill*:

- `vars`
- `split`
- `min_level`
- `max_level`
- `ql`
- `qr`
- `vcenter`
- `cmap`

More explanations of these arguments are given in *Create isofill/contourf plots*.

The *base_utils.Pcolor* class shares the same signature as *base_utils.Boxfill*, and their usages are also identical. (Honestly, is there any difference between the two?)

5.4.2 Basic plot example

A boxfill/imshow/pcolormesh plot is also relatively easier to create. See a simple example below:

```
import matplotlib.pyplot as plt
import gplot
from gplot.lib import netcdf4_utils

var = netcdf4_utils.readData('msl')
lats = netcdf4_utils.readData('latitude')
lons = netcdf4_utils.readData('longitude')

figure, axes = plt.subplots( figsize=(12, 6), nrows=1, ncols=2,
        constrained_layout=True)
box = gplot.Boxfill(var1)
pc = gplot.Pcolor(var1)
gplot.plot2( var1, box, axes[0], title='default Boxfill', projection='cyl',
        legend='local')
gplot.plot2( var1, pc, axes[1], title='default Pcolor', projection='cyl',
        legend='local')
figure.show()
```

The result is given in Fig.5.5 below:

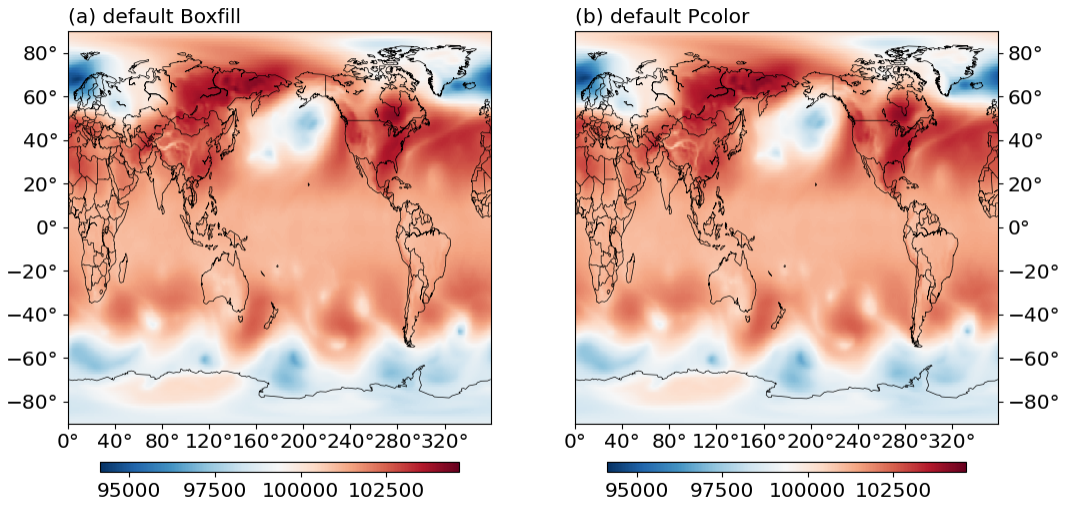

Fig. 5.5: Boxfill (a) and Pcolor (b) plot examples.

5.4.3 The mappable object

Same as an isofill/isoline plot, the *mappable object* of a boxfill/imshow/pcolormesh plot is stored as an attribute of the `base_utils.Plot2D` (or `basemap_utils.Plot2Basemap`) object. See *The mappable object*.

5.5 Add colorbars to plots

Table of Contents

5.5.1 Positioning of the colorbar

A colorbar is automatically added for an *Isofill/contourf*, a *Boxfill/imshow* and a polychromatic *Isoline/contour* plot.

The positioning of the colorbar is controlled by the `legend` keyword argument to the `base_utils.Plot2D.__init__()`, or the `base_utils.plot2()` function. It can have 1 of the 3 possible values:

- `legend = 'global'`: the default. If there are more than 1 subplots in the figure, all subplots share the same colorbar, which is created by the 1st subplot. If only 1 subplot in the figure, same as `legend = 'local'`.
- `legend = 'local'`: subplots in the figure have their own colorbars.
- `legend = None`: don't plot the colorbar.

Note: *gplot* at the moment does not support colorbars that are shared by a subset of the subplots, like in the examples given in the matplotlib tutorial.

Additionally, the `legend_ori` argument specifies the orientation of the colorbar:

- `legend_ori = 'horizontal'`: horizontal colorbar.
- `legend_ori = 'vertical'`: vertical colorbar.

Note: Only the right (for vertical colorbar) and bottom (for horizontal colorbar) side of the subplot/figure placement are supported. Top and left side placement are not supported in *gplot*. However, one can create the colorbar on their own, by setting `legend = None`, and using the returned *mappable object*.

5.5.2 Overflows on a colorbar

Overflow is represented by a triangle on either end of the colorbar (see Fig.5.6 below for an example). It signals that all values below the minimum overflow level are represented by the color of the left triangle, and all values above the maximum overflow level by the right triangle. Namely, one chooses to selectively plot only a sub-range of the data values.

Overflows are **ONLY** added if the range of data exceeds the range plotted. And they can be introduced by setting the `min_level` or `ql` arguments (for the left overflow), and the `max_level` or `qr` arguments (for the right overflow).

See also:

`base_utils.Isofill`, *`base_utils.Isoline`*, *`base_utils.Boxfill`*

5.5.3 Alternating top and bottom ticks of a horizontal colorbar

In an *Isofill/contourf* plot, if the number of levels is too big, the tick labels of a horizontal colorbar may start to overlap with each other. In some cases this can be solved by putting half of the tick labels on the top side and half on the bottom side (see Fig.5.6 above or *this figure* for examples).

This functionality is automatically enabled, but only for *Isofill/contourf* plots with horizontal colorbars.

Note: If the number of contour levels keeps on growing, the tick labels may start to overlap again. In such cases, it is worth trying either reducing the level numbers, or using a smaller font size.

5.6 Create quiver plots

Table of Contents

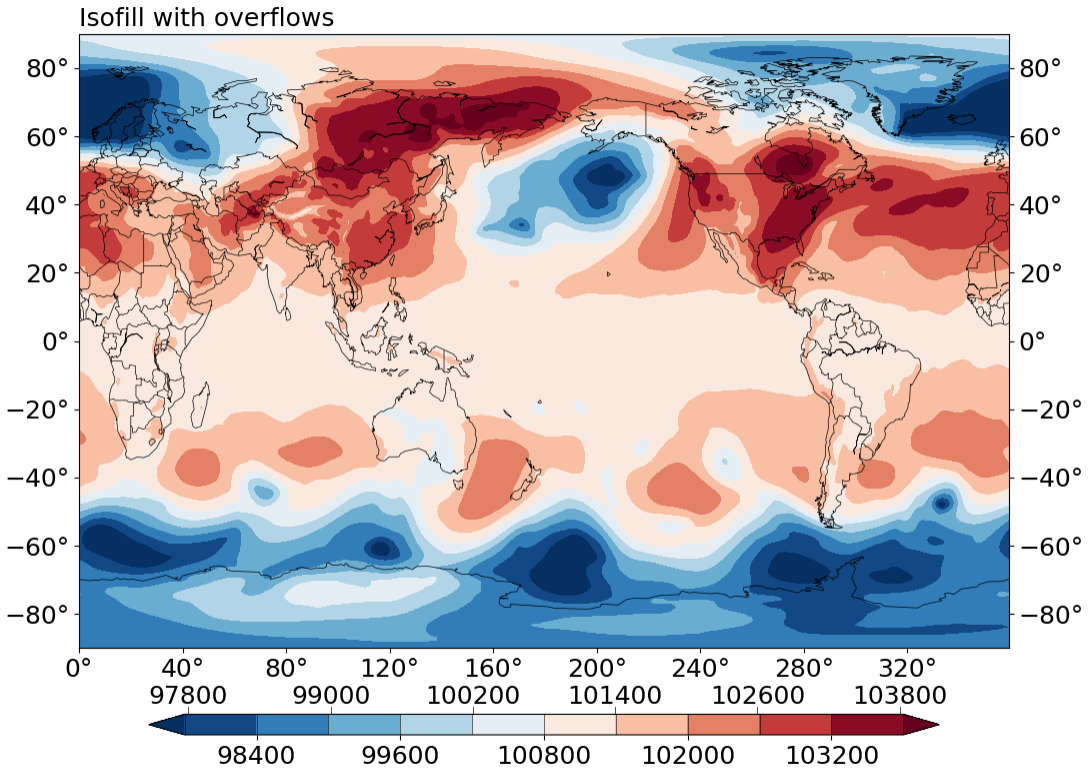

Fig. 5.6: Isofill plot with overflows on both sides.

5.6.1 The `Quiver` class

To create a 2D quiver plot, one creates a *base_utils.Quiver* object as the plotting method, and passes it to the *base_utils.Plot2Quiver* constructor or the *base_utils.plot2()* function.

The `__init__()` of *base_utils.Quiver* takes these input arguments:

- `step`
- `reso`
- `scale`
- `keylength`
- `linewidth`
- `color`
- `alpha`

`linewidth`, `color` and `alpha` should be self-explanatory. Others are explained in further details below.

5.6.2 Control the quiver density

When the input data have too fine a resolution, the quiver plot may end up being too dense and not quite readable (see Fig.5.7a below for an example). This can be solved by either

1. sub-sampling the data with a step: `u = u[::step, ::step]; v = v[::step, ::step]`, or
2. regridding the data to a lower resolution `reso`.

Method 1 is controlled by the `step` input argument (see Fig.5.7b below for an example), and the latter method the `reso` argument (see Fig.5.7c,d). If both are given, the latter one takes precedence.

Note: regridding requires *scipy* as an optional dependency.

5.6.3 Control the quiver lengths

The lengths of the quiver arrows are controlled by the `scale` argument. A larger scale value creates shorter arrows. When left as the default `None`, it will try to derive a suitable scale level for the given inputs.

The length of the reference quiver arrow is controlled by the `keylength` argument. Given a set `scale`, a larger `keylength` makes the **reference** quiver arrow longer. Similar as `scale`, `keylength` is default to `None`, and the plotting function will try to derive a suitable value automatically for you.

Fig.5.8 below shows some examples of controlling the lengths.

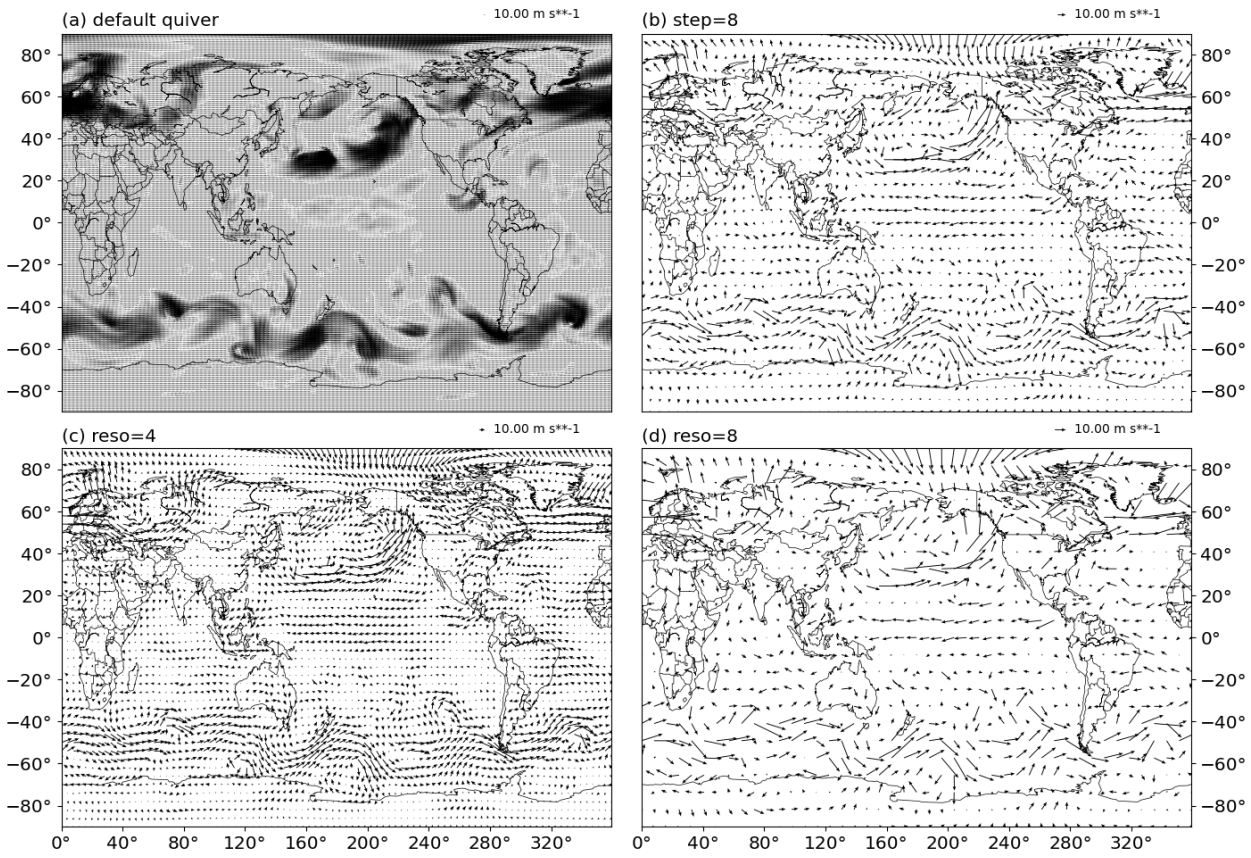

Fig. 5.7: Density control of a quiver plot. (a) default quiver density `q = Quiver()`. (b) reduced density by subsampling: `q = Quiver(step=8)`. (c) reduced density by regridding: `q = Quiver(reso=4)`. (d) reduced density by regridding: `q = Quiver(reso=8)`.

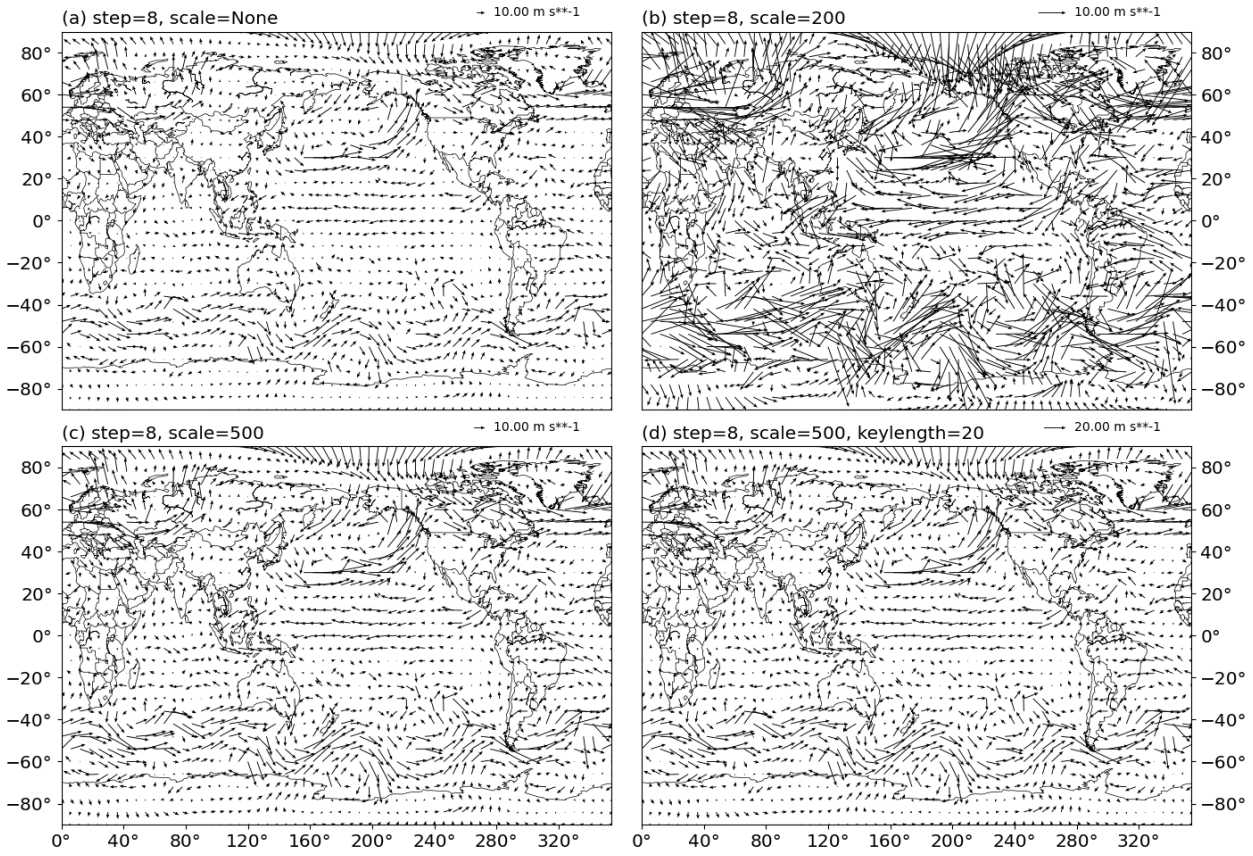

Fig. 5.8: Length control of a quiver plot. (a) automatic scale `q = Quiver(step=8, scale=None)`. (b) specify scale=200: `q = Quiver(step=8, scale=200)`. (c) specify scale=500: `q = Quiver(step=8, scale=500)`. (d) specify scale=500, keylength=20: `q = Quiver(step=8, scale=500, keylength=20)`.

5.6.4 Quiver overlay

It is common to see quiver plots superimposed on top of an isofill/contourf plot. To achieve this, simply re-use the same `axis` object in the isofill/contourf plot, and the subsequent quiver plot. E.g.

```
figure = plt.figure(figsize=(12, 10), dpi=100)
ax = figure.add_subplot(111)
iso = gplot.Isofill(var1)
q = gplot.Quiver(reso=5, scale=500)

gplot.plot2(var1, iso, ax, projection='cyl')
gplot.plot2(u, q, var_v=v, xarray=lons, yarray=lats,
            ax=ax, title='quiver overlay', projection='cyl')
figure.show()
```

The result is given in Fig.5.9 below.

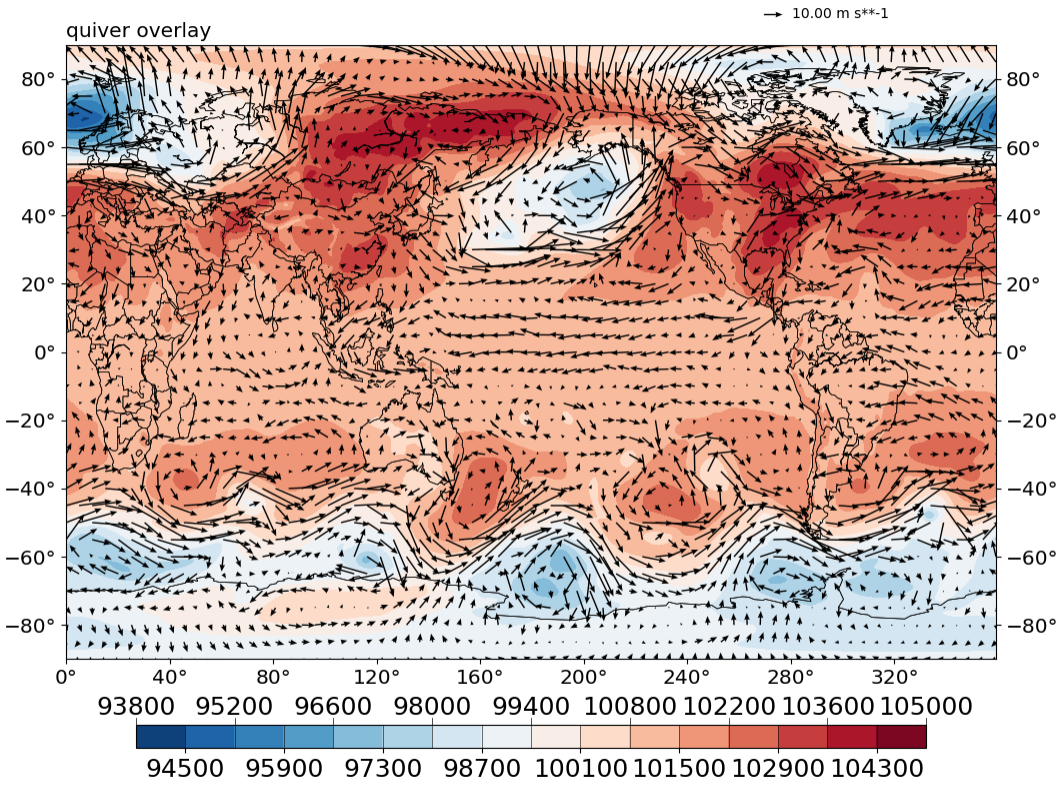

Fig. 5.9: Quiver plot on top of isofill.

5.6.5 Curved quiver plots

Sometimes one needs to visualize a vector field in a region where the vector magnitudes are rather small, and a larger domain is needed to be shown at the same time to give enough context. In such cases, when the `scale` is adjusted to a comfortable value for the target region to be readable, other regions may have quiver arrows that are too large and the plot looks messy.

One possible solution is to use curved quivers rather than straight ones. `matplotlib` does not support this out-of-the-box, some hacks are used to achieve this. Due credits to the author of this repo, and this stackoverflow answer.

A curved quiver plot is done by passing in `curve=True`, e.g.:

```
figure = plt.figure(figsize=(12, 10), dpi=100)
ax = figure.add_subplot(111)
q = gplot.Quiver(step=8)
pquiver = Plot2QuiverBasemap(
    u, v, q, xarray=lons, yarray=lats, ax=ax, title='curved quiver',
    projection='cyl', curve=True)
pquiver.plot()

figure.show()
```

The result is given in Fig.5.10 below.

Note: Curved quiver plot takes notably longer to generate, and is considered experimental at the moment.

5.6.6 The mappable object

The *mappable object* of a quiver plot is stored as an attribute of the *base_utils.Plot2Quiver* (or *basemap_utils.Plot2QuiverBasemap*) object:

```
>>> q = gplot.Quiver()
>>> pobj = Plot2QuiverBasemap(u, v, q, xarray=lons, yarray=lats, ax=ax, projection='cyl')
>>> pobj.plot()
>>> pobj.quiver
<matplotlib.quiver.Quiver object at 0x7f2e03aed750>
```

5.7 Managing subplots

Table of Contents

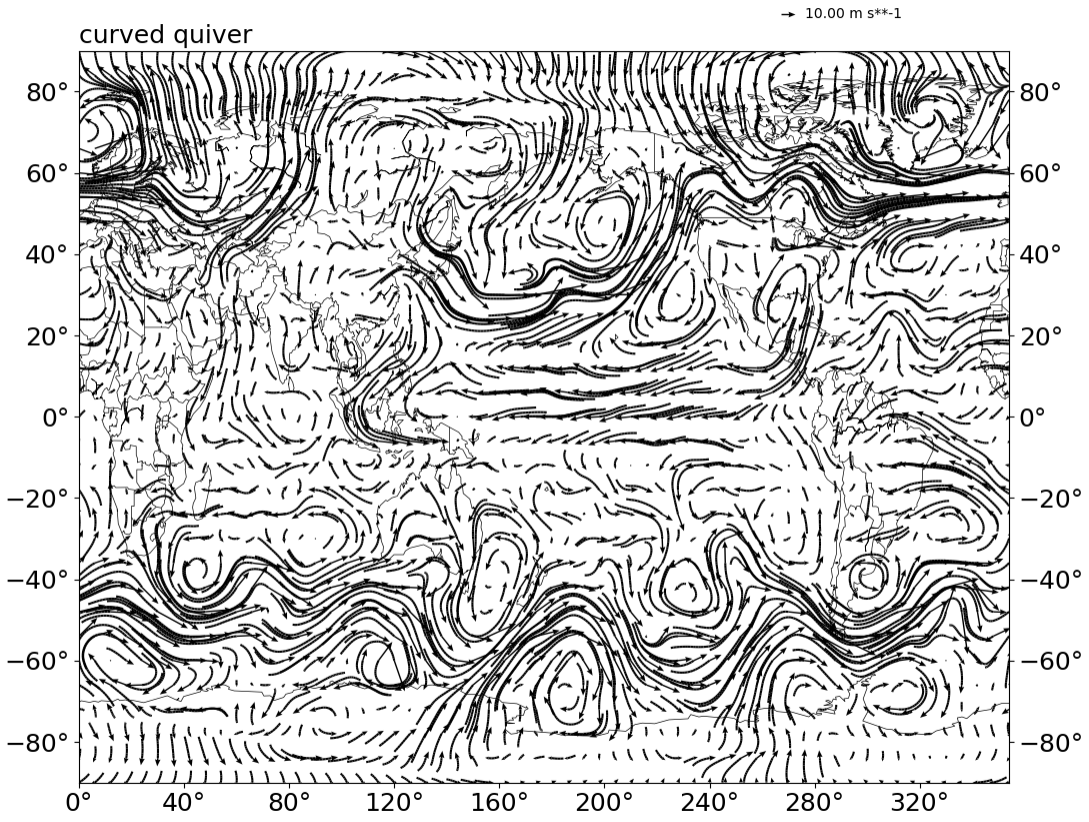

Fig. 5.10: Curved quiver plot.

5.7.1 Recommended way of creating subplots

In academic works, people usually compose a single figure with multiple subplots, sometimes to facilitate comparisons, but mostly to make the most of the valuable real estates of a graph.

There are more than one ways of creating subplots in `matplotlib`. For usage with `gplot`, the recommended way of creating subplots is:

```
figure, axes = plt.subplots(nrows=2, ncols=2, constrained_layout=True)
```

The returned `axes` is a 2D array, holding the axes for a 2x2 grid layout. To iterate through the axes, one can use:

```
for ii, axii in enumerate(axes.flat):
    rowii, colii = np.unravel_index(ii, (nrows, ncols))
    ...
```

Note: the `constrained_layout=True` argument is recommended. This will adjust the spacings of the subplots to avoid overlaps between subplots, and wasted spaces as well. **Do not** use `figure.tight_layout()` afterwards, as it tends to mess up the placement of a shared, global colorbar.

Note: the placement of a globally shared colorbar is currently not as robust as a local colorbar. One may find the global colorbar tick labels overlaping with those in the bottom row x-axis, if the `constrained_layout` is not set to `True`.

5.7.2 Automatically label the subplots with alphabetic indices

The `title` input argument to the *base_utils.Plot2D* constructor or the *base_utils.plot2()* function is used to label the subplots. It is defaulted to `None`. The `clean` argument also has some effects. They function a bit differently in different scenarios:

- `title = None`:
  - If figure has only 1 subplot: no title is drawn.
  - If figure has more than 1 subplots, an alphabetic index is used as the subplot title, e.g. `(a)` for the 1st subplot, `(b)` for the 2nd, and so on. The order is row-major. After using up all the 26 letters, it will cycle through them again but with 2 letters at a time, e.g. `(aa)` for the 27th subplots. This rarely happens in practice.
- `title = some_text`:
  - If figure has only 1 subplot: use `some_text` as the title.
  - If figure has more than 1 subplots, an alphabetic index is prepended to form the subplot title: `(a) some_text`. An example of this can be seen *here*.
- `title = (x) some_text`:

  Where `x` is an arbitrary string. Use `(x) some_text` as the title. This can be used to override the automatic row-major ordering of the subplot indices. For instance, you want to label it as `(k)` when the subplot is at a position of `(h)`.
- `title = 'none'` or `clean = True`: no title is drawn in any circumstances.

5.8 Other asepcts

Table of Contents

5.8.1 netCDF interfaces

The *base_utils.Plot2D*, `base_utils.Plot2Basemap` and `base_utils.Plot2Cartopy` classes (and their derived classes, *base_utils.Plot2Quiver*, `base_utils.Plot2QuiverBasemap` and `base_utils.Plot2QuiverCartopy`) all expect plain `ndarray` as input data. However, the *base_utils.plot2()* interface function can accept other data types. E.g. the *netCDF* data read in by *CDAT* is a `TransientVariable` object, which is a derived type of `np.ma.MaskedArray`, and carries the metadata with it. Other netCDF file I/O modules, like *Iris* and *Xarray* also provide their own data types. The `nc_interface` argument to the *base_utils.plot2()* function tells the function which module has been used in reading in the *netCDF* data, and some preprocessing can be done accordingly to retrieve some necessary information, including the x- and y- coordinates, data units etc..

`nc_interface` can be one of these:

- `netcdf`
- `cdat`
- `iris`
- `xarray`

Note: Currently, only `netcdf` and `cdat` are supported.

5.8.2 Axes ticks and ticklabels

The axes ticks and ticklabels are controlled by the `label_axes` keyword argument to the `__init__` method of *base_utils.Plot2D* and *base_utils.plot2()*. It is defaulted to `True`. The `clean` keyword argument also has some effects.

The different values of `label_axes` are:

- `True`: default.
  - If figure has only 1 subplot, default to plot the left, bottom and right hand side axes ticks and ticklabels.
  - If figure has more than 1 subplots, default to plot only the exterior facing (except for the top side) axes ticks and ticklabels. E.g. in a 2x2 subplot layout, the top-left subplot has only the left axes ticks/ticklabels, the bottom-right subplot only the right and bottom axes ticks/ticklabels, etc.. See *this figure* for an example. This is the same as the `sharex` and `sharey` options in `plt.subplots(sharex=True, sharey=True)`.
- `False`: turn off axes ticks/ticklabels on all sides.

- `'all'`: turn on axes ticks/ticklabels on all sides.
- `(left, right, top, bottom)`: a 4 boolean element tuple, specifying the left, right, top and bottom side axes ticks/ticklabels. See Fig.5.11 for an example.

Note: Setting `clean=True` also turns off axes ticks/tickslabels on all sides.

Note: Notice that in Fig.5.11, when the bottom side axes ticklabels are turned off, the spacing between bottom axis and colorbar also adjusts so as to avoid leaving a wasted space.

Additionally, setting `axes_grid = True` will add axis grid lines. This is turned off by default, and is independent from the axis ticks/ticklabels: one can have only axes grid lines without any ticks/ticklabels.

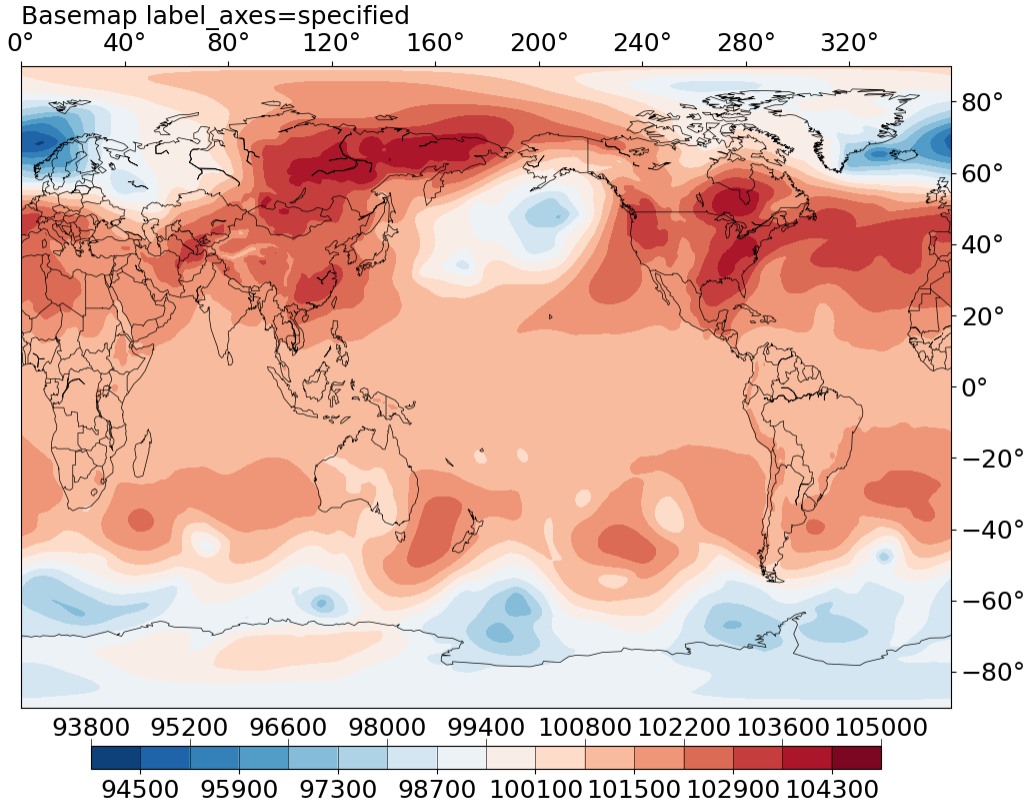

Fig. 5.11: Specify the axis ticks/ticklabels by setting `label_axes = (0, 1, 1, 0)`. The 4 elements in the tuple correspond to the left, right, top, bottom sides, respectively.

5.8.3 Color for missing values

If not set, `matplotlib` sets the default background color to white, which also appears in many colormaps (e.g. the `plt.cm.RdBu_r` used as default colormap of *gplot*). Therefore it is easy to confuse your audience with the missing values and valid data values that happen to be represented with white color (or something very close to white). See the comparison below:

Therefore, to avoid such ambiguities, the missing values are represented by `fill_color` in *gplot*, using:

```
self.ax.patch.set_color(self.fill_color)
```

where `fill_color` is a keyword argument to the `__init__` method of *base_utils.Plot2D* and *base_utils.plot2()*. It is defaulted to a grey color (`0.8`).

5.8.4 Font size

The font sizes are controlled by the `fontsize` keyword argument to the `__init__` method of *base_utils.Plot2D* and *base_utils.plot2()*. It is defaulted to 11, and affects the sizes of these texts in a plot:

- title
- axes ticklabels
- axes labels
- colorbar ticklabels and units
- reference quiver key units

When the figure has more than 1 subplots, the font sizes are adjusted by the following empirical formula:

$$s_{adj} = \frac{7}{MAX\{n_r, n_c\}} + s_0$$

where:

- s_0 is the `fontsize` argument (default to 11).
- n_r, n_c: the number of rows, columns in the subplot layout.
- s_{adj}: the adjusted font size for the subplot.

5.8.5 Default parameters

gplot defines the following dictionary of default parameters:

```
# Default parameters
rcParams = {
    'legend': 'global',
    'title': None,
    'label_axes': True,
    'axes_grid': False,
    'fill_color': '0.8',
    'projection': 'cyl',
    'legend_ori': 'horizontal',
    'clean': False,
    'bmap': None,
```

(continues on next page)

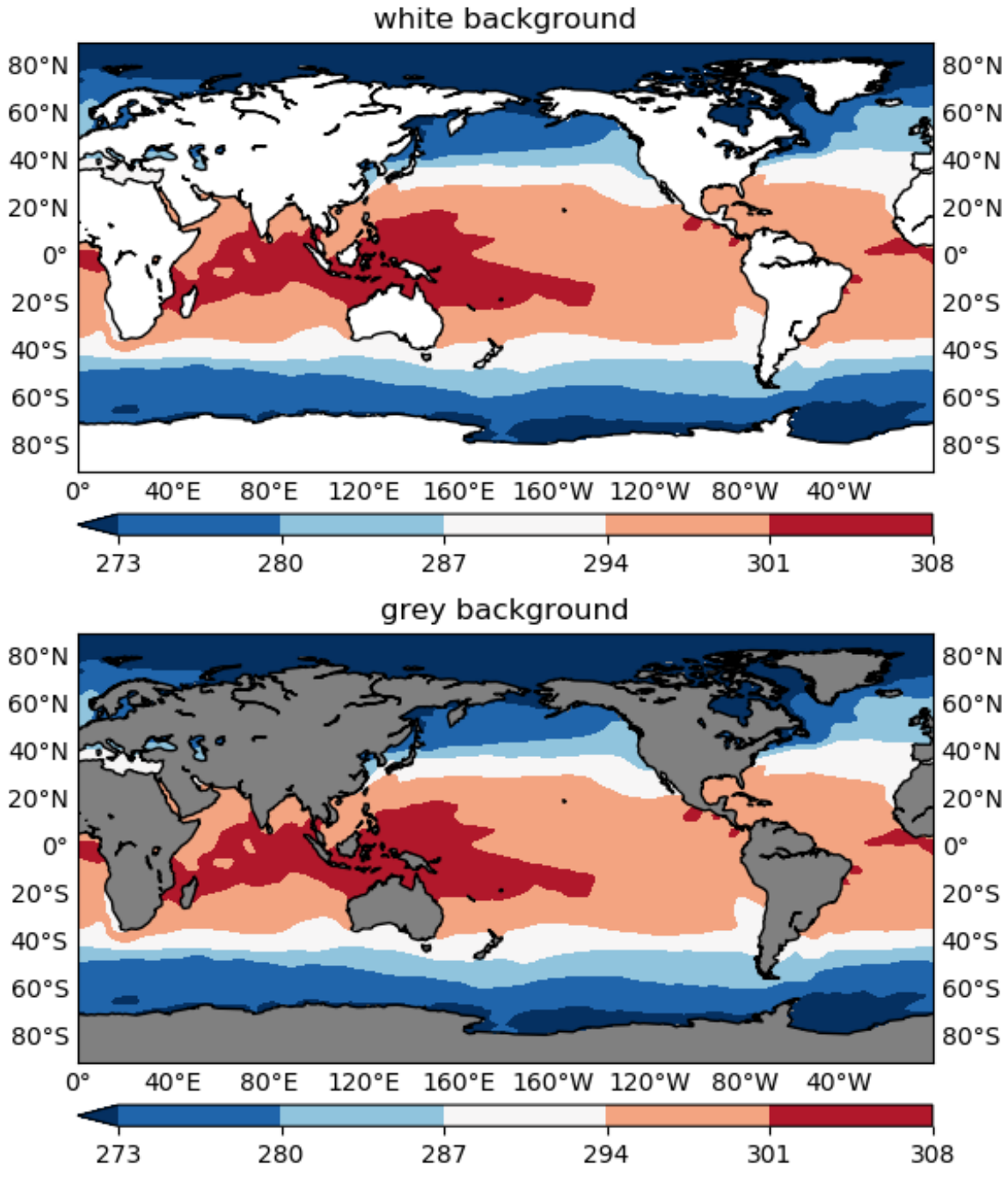

Fig. 5.12: Comparison of the missing values as represented with a white background (top) and grey background (bottom).

(continued from previous page)

```
    'isgeomap': True,
    'fix_aspect': False,
    'nc_interface': 'cdat',
    'geo_interface': 'basemap',
    'fontsize': 11,
    'verbose': True,
    'default_cmap': plt.cm.RdBu_r
}
```

The `base_utils.rcParams` dict can be altered to make a change persistent in a Python session. And the *base_utils.restoreParams()* can be used to restore the original values. E.g.

```
gplot.rcParams['fontsize'] = 4

test_basemap_default()
test_basemap_isofill_overflow()

gplot.restoreParams()

test_basemap_isolines()
```

CHAPTER

SIX

GPLOT MODULE CONTENTS

6.1 Documentation page for base_utils.py

Basic 2D plotting functions and classes.

Contains:

- utility functions.
- plotting method classes.
- plotting wrapper classes, from which equivalent geographical plotting classes are inherited.

Memebers in this module are available under the *gplot* namespace:

gplot.xxx

Author: guangzhi XU (xugzhi1987@gmail.com) Update time: 2021-02-13 10:06:58.

class base_utils.**Boxfill**(*vars*, *split=2*, *min_level=None*, *max_level=None*, *ql=None*, *qr=None*, *vcenter=0*, *cmap=None*, *verbose=True*)

Plotting method for boxfill/imshow plots

__init__(*vars*, *split=2*, *min_level=None*, *max_level=None*, *ql=None*, *qr=None*, *vcenter=0*, *cmap=None*, *verbose=True*)

Plotting method for boxfill/imshow plots

Parameters

vars (*ndarray or list*) – if ndarray, input data to create 2d plot from. If list, a list of ndarrays.

Keyword Arguments

- **split** (*int*) – whether to split the colormap at a given value (<vcenter>) into 2 parts or not. Can be 1 of these 3 values: 0: do not split. 1: split at <vcenter> only if range of data in <vars> strides

  <vcenter>.

  2: force split at <vcenter>. If split and data range strides across <vcenter>, will use the lower half of the colormap for values <= <vcenter>, the upper half of the colormap for values >= <vcenter>. If split and data range on 1 side of <vcenter>, will only use only half of the colormap range, depending on whether data are on which side of <vcenter>.

- **levels** (*list, tuple or 1darray*) – specified contour levels. If not given, compute contour levels using <num>, <zero>, <min_level>, <max_level>, <ql>, <qr>.

- **min_level** (*float or None*) – specified minimum level to plot. If None, determine from <ql> if given. If both <min_level> and <ql> are None, use minimum value from <vars>. If both given, take the larger.
- **max_level** (*float or None*) – specified maximum level to plot. If None, determine from <qr> if given. If both <max_level> and <qr> are None, use maximum value from <vars>. If both given, take the smaller.
- **ql** (*float or None*) – specified minimum left quantile to plot. If None, determine from <min_level> if given. If both <min_level> and <ql> are None, use minimum value from <vars>. If both given, take the larger.
- **qr** (*float or None*) – specified maximum right quantile (e.g. 0.01 for the 99th percentile) to plot. If None, determine from <max_level> if given. If both <max_level> and <qr> are None, use maximum value from <vars>. If both given, take the smaller.
- **vcenter** (*float*) – value at which to split the colormap. Default to 0.
- **cmap** (*matplotlib colormap or None*) – colormap to use. If None, use the default in rcParams['default_cmap'].
- **verbose** (*bool*) – whether to print some info or not.

class base_utils.**GIS**(*xpixels=2000*, *dpi=96*, *verbose=True*)

Plotting method for GIS plots

__init__(*xpixels=2000*, *dpi=96*, *verbose=True*)

Plotting method for GIS plots

Keyword Arguments

- **xpixels** (*int*) – plot size.
- **dpi** (*int*) – dpi.
- **verbose** (*bool*) – whats this?

class base_utils.**Hatch**(*hatch='.'*, *color='k'*, *alpha=1.0*)

Plotting method for hatching plots

__init__(*hatch='.'*, *color='k'*, *alpha=1.0*)

Plotting method for hatching plots

Keyword Arguments

- **hatch** (*str*) – style of hatching. Choices:
- **'0'** ('.', '/', '//', '', '\', '*', '-', '+', 'x', 'o',) –
- **alpha** (*float*) – transparent level, in range of [0, 1].

class base_utils.**Isofill**(*vars*, *num=15*, *zero=1*, *split=1*, *levels=None*, *min_level=None*, *max_level=None*, *ql=None*, *qr=None*, *vcenter=0*, *cmap=None*, *stroke=False*, *stroke_color='0.3'*, *stroke_lw=0.2*, *stroke_linestyle='-'*, *verbose=True*)

Plotting method for isofill/contourf plots

__init__(*vars*, *num=15*, *zero=1*, *split=1*, *levels=None*, *min_level=None*, *max_level=None*, *ql=None*, *qr=None*, *vcenter=0*, *cmap=None*, *stroke=False*, *stroke_color='0.3'*, *stroke_lw=0.2*, *stroke_linestyle='-'*, *verbose=True*)

Plotting method for isofill/contourf plots

Parameters

vars (*ndarray or list*) – if ndarray, input data to create 2d plot from. If list, a list of ndarrays.

Keyword Arguments

- **num** (*int*) – the desired number of contour levels. NOTE that the resultant number may be slightly different.
- **zero** (*int*) – whether 0 is allowed to be a contour level. -1 for not allowed, 0 or 1 otherwise.
- **split** (*int*) – whether to split the colormap at a given value (<vcenter>) into 2 parts or not. Can be 1 of these 3 values: 0: do not split. 1: split at <vcenter> only if range of data in <vars> strides

  <vcenter>.

  2: force split at <vcenter>. If split and data range strides across <vcenter>, will use the lower half of the colormap for values <= <vcenter>, the upper half of the colormap for values >= <vcenter>. If split and data range on 1 side of <vcenter>, will only use only half of the colormap range, depending on whether data are on which side of <vcenter>.
- **levels** (*list, tuple or 1darray*) – specified contour levels. If not given, compute contour levels using <num>, <zero>, <min_level>, <max_level>, <ql>, <qr>.
- **min_level** (*float or None*) – specified minimum level to plot. If None, determine from <ql> if given. If both <min_level> and <ql> are None, use minimum value from <vars>. If both given, take the larger.
- **max_level** (*float or None*) – specified maximum level to plot. If None, determine from <qr> if given. If both <max_level> and <qr> are None, use maximum value from <vars>. If both given, take the smaller.
- **ql** (*float or None*) – specified minimum left quantile to plot. If None, determine from <min_level> if given. If both <min_level> and <ql> are None, use minimum value from <vars>. If both given, take the larger.
- **qr** (*float or None*) – specified maximum right quantile (e.g. 0.01 for the 99th percentile) to plot. If None, determine from <max_level> if given. If both <max_level> and <qr> are None, use maximum value from <vars>. If both given, take the smaller.
- **vcenter** (*float*) – value at which to split the colormap. Default to 0.
- **cmap** (*matplotlib colormap or None*) – colormap to use. If None, use the default in rcParams['default_cmap'].
- **stroke** (*bool*) – whether to overlay a layer of thin contour lines on top of contourf.
- **stroke_color** (*str or color tuple*) – color to plot the overlying thin contour lines.
- **stroke_lw** (*float*) – line width to plot the overlying thin contour lines.
- **stroke_linestyle** (*str*) – line style to plot the overlying thin contour lines.
- **verbose** (*bool*) – whether to print some info or not.

class base_utils.**Isoline**(*vars*, *num=15*, *zero=1*, *split=1*, *levels=None*, *min_level=None*, *max_level=None*, *ql=None*, *qr=None*, *vcenter=0*, *cmap=None*, *black=False*, *color=None*, *linewidth=1.0*, *alpha=1.0*, *dash_negative=True*, *bold_lines=None*, *label=False*, *label_fmt=None*, *label_box=False*, *label_box_color='w'*, *verbose=True*)

Plotting method for isoline/contour plots

__init__(*vars*, *num=15*, *zero=1*, *split=1*, *levels=None*, *min_level=None*, *max_level=None*, *ql=None*, *qr=None*, *vcenter=0*, *cmap=None*, *black=False*, *color=None*, *linewidth=1.0*, *alpha=1.0*, *dash_negative=True*, *bold_lines=None*, *label=False*, *label_fmt=None*, *label_box=False*, *label_box_color='w'*, *verbose=True*)

Plotting method for isoline/contour plots

Parameters

vars (*ndarray or list*) – if ndarray, input data to create 2d plot from. If list, a list of ndarrays.

Keyword Arguments

- **num** (*int*) – the desired number of contour levels. NOTE that the resultant number may be slightly different.
- **zero** (*int*) – whether 0 is allowed to be a contour level. -1 for not allowed, 0 or 1 otherwise.
- **split** (*int*) – whether to split the colormap at a given value (<vcenter>) into 2 parts or not. Can be 1 of these 3 values: 0: do not split. 1: split at <vcenter> only if range of data in <vars> strides

  <vcenter>.

  2: force split at <vcenter>. If split and data range strides across <vcenter>, will use the lower half of the colormap for values <= <vcenter>, the upper half of the colormap for values >= <vcenter>. If split and data range on 1 side of <vcenter>, will only use only half of the colormap range, depending on whether data are on which side of <vcenter>.
- **levels** (*list, tuple or 1darray*) – specified contour levels. If not given, compute contour levels using <num>, <zero>, <min_level>, <max_level>, <ql>, <qr>.
- **min_level** (*float or None*) – specified minimum level to plot. If None, determine from <ql> if given. If both <min_level> and <ql> are None, use minimum value from <vars>. If both given, take the larger.
- **max_level** (*float or None*) – specified maximum level to plot. If None, determine from <qr> if given. If both <max_level> and <qr> are None, use maximum value from <vars>. If both given, take the smaller.
- **ql** (*float or None*) – specified minimum left quantile to plot. If None, determine from <min_level> if given. If both <min_level> and <ql> are None, use minimum value from <vars>. If both given, take the larger.
- **qr** (*float or None*) – specified maximum right quantile (e.g. 0.01 for the 99th percentile) to plot. If None, determine from <max_level> if given. If both <max_level> and <qr> are None, use maximum value from <vars>. If both given, take the smaller.
- **vcenter** (*float*) – value at which to split the colormap. Default to 0.
- **cmap** (*matplotlib colormap or None*) – colormap to use. If None, use the default in rcParams['default_cmap'].
- **black** (*bool*) – use black lines instead of colored lines.
- **color** (*str or color tuple*) – color to plot the contour lines.
- **linewidth** (*float*) – line width to plot the contour lines.
- **alpha** (*float*) – transparent level, in range of [0, 1].
- **dash_negative** (*bool*) – whether to use dashed lines for negative contours.
- **bols_lines** (*list if None*) – if a list, values to highlight using bold lines (line width scaled by 2.0).

- **label** (*bool*) – whether to label the contour lines or not.
- **label_fmt** (*str or dict or None*) – if <label> is True, format string to format contour levels. E.g. '%0.2f'. If None, automatically derive a format suitable for the contour levels.
- **label_box** (*bool*) – whether to put contour labels in a bounding box with background color or not.
- **label_box_color** (*str or color tuple*) – if <label_box> is True, the background color for the bounding boxes for the labels.
- **verbose** (*bool*) – whether to print some info or not.

class base_utils.**Pcolor**(*vars*, *split=2*, *min_level=None*, *max_level=None*, *ql=None*, *qr=None*, *vcenter=0*, *cmap=None*, *verbose=True*)

Plotting method for pcolormesh plots

__init__(*vars*, *split=2*, *min_level=None*, *max_level=None*, *ql=None*, *qr=None*, *vcenter=0*, *cmap=None*, *verbose=True*)

Plotting method for pcolormesh plots

Parameters

vars (*ndarray or list*) – if ndarray, input data to create 2d plot from. If list, a list of ndarrays.

Keyword Arguments

- **split** (*int*) – whether to split the colormap at a given value (<vcenter>) into 2 parts or not. Can be 1 of these 3 values: 0: do not split. 1: split at <vcenter> only if range of data in <vars> strides

  <vcenter>.

  2: force split at <vcenter>. If split and data range strides across <vcenter>, will use the lower half of the colormap for values <= <vcenter>, the upper half of the colormap for values >= <vcenter>. If split and data range on 1 side of <vcenter>, will only use only half of the colormap range, depending on whether data are on which side of <vcenter>.
- **levels** (*list, tuple or 1darray*) – specified contour levels. If not given, compute contour levels using <num>, <zero>, <min_level>, <max_level>, <ql>, <qr>.
- **min_level** (*float or None*) – specified minimum level to plot. If None, determine from <ql> if given. If both <min_level> and <ql> are None, use minimum value from <vars>. If both given, take the larger.
- **max_level** (*float or None*) – specified maximum level to plot. If None, determine from <qr> if given. If both <max_level> and <qr> are None, use maximum value from <vars>. If both given, take the smaller.
- **ql** (*float or None*) – specified minimum left quantile to plot. If None, determine from <min_level> if given. If both <min_level> and <ql> are None, use minimum value from <vars>. If both given, take the larger.
- **qr** (*float or None*) – specified maximum right quantile (e.g. 0.01 for the 99th percentile) to plot. If None, determine from <max_level> if given. If both <max_level> and <qr> are None, use maximum value from <vars>. If both given, take the smaller.
- **vcenter** (*float*) – value at which to split the colormap. Default to 0.
- **cmap** (*matplotlib colormap or None*) – colormap to use. If None, use the default in rcParams['default_cmap'].
- **verbose** (*bool*) – whether to print some info or not.

class base_utils.**Plot2D**(*var*, *method*, *ax=None*, *xarray=None*, *yarray=None*, *title=None*, *label_axes=True*, *axes_grid=False*, *legend='global'*, *legend_ori='horizontal'*, *clean=False*, *fontsize=None*, *fill_color=None*)

Base 2D plotting class

For geographical plots, see Plot2Basemap or Plot2Cartopy, which handles equivalent plotting with geographical map projections.

__init__(*var*, *method*, *ax=None*, *xarray=None*, *yarray=None*, *title=None*, *label_axes=True*, *axes_grid=False*, *legend='global'*, *legend_ori='horizontal'*, *clean=False*, *fontsize=None*, *fill_color=None*)

Parameters

- **var** (*ndarray*) – input data to plot. Determines what to plot. Mush have dimensions >= 2. For data with rank>2, take the slab from the last 2 dimensions.
- **method** (*PlotMethod*) – plotting method. Determines how to plot. Could be Isofill, Isoline, Boxfill, Quiver, Shading, Hatch, GIS.

Keyword Arguments

- **ax** (*matplotlib axis or None*) – axis obj. Determines where to plot. If None, create a new.
- **xarray** (*1darray or None*) – array to use as the x-coordinates. If None, use the indices of the last dimension: np.arange(slab.shape[-1]).
- **yarray** (*1darray or None*) – array to use as the y-coordinates. If None, use the indices of the 2nd last dimension: np.arange(slab.shape[-2]).
- **title** (*str or None*) – text as the figure title if <ax> is the single plot in the figure. If None, automatically get an alphabetic subtitle if <ax> is a subplot, e.g. '(a)' for the 1st subplot, '(d)' for the 4th one. If str and <ax> is a subplot, prepend <title> with the alphabetic index. One can force overriding the alphabetic index by giving a title str in the format of '(x) xxxx', e.g. '(p) subplot-p'.
- **label_axes** (*bool or 'all' or tuple*) – controls axis ticks and ticklabels. If True, don't exert any inference other than changing the ticklabel fontsize, and let matplotlib put the ticks and ticklabels (i.e. default only left and bottom axes). If False, turn off all ticks and ticklabels. If 'all', plot ticks and ticks labels on all 4 sides. If (left, right, top, bottom), specify which side to plot ticks/ticklabels. Each swith is a bool or binary. If None, will set the ticks/ticklabels such that the interior subplots have no ticks/ticklabels, edge subplots have ticks/ticklabels on the outer edges, i.e. similar as the 'sharex', 'sharey' options. Location of the subplot is determined from return of *ax.get_geometry()*.
- **axes_grid** (*bool*) – whether to add axis grid lines.
- **legend** (*str or None*) – controls whether to share colorbar or not. A colorbar is only plotted for Isofill/Isoline plots. If None, don't put colorbar. If 'local', <ax> has its own colorbar. If 'global', all subplots in the figure share a single colorbar, which is created by the 1st subplot in the figure, which is determined from the return of *ax.get_geometry()*.
- **legend_ori** (*str*) – orientation of colorbar. 'horizontal' or 'vertical'.
- **clean** (*bool*) – if False, don't plot axis ticks/ticklabels, colorbar, axis grid lines or title.
- **fontsize** (*int*) – font size for ticklabels, title, axis labels, colorbar ticklabels.
- **fill_color** (*str or color tuple*) – color to use as background color. If data have missings, they will be shown as this color. It is better to use a grey than while to better distinguish missings.

`alternateTicks`(*cbar*, *ticks*)

Create alternating ticks and ticklabels for colorbar

Parameters

- **cbar** (*matplotlib colorbar obj*) – input colorbar obj to alter.
- **ticks** (*list or array*) – ticks of the colorbar.

Returns

cbar (matplotlib colorbar obj) – the altered colorbar.

Only works for horizontal colorbar with discrete ticks. As vertical colorbar doesn't tend to have overlapping tick labels issue.

Update time: 2021-12-30 11:07:10: deprecated, use global function alternateTicks() instead.

`classmethod getExtend`(*method*)

Get colorbar overflow on both ends

Returns

extend (str) –

'both', 'min', 'max' or 'neither'. Determined
from the method obj.

`getGeo`()

Get geometry layout of the axis and font size

Returns

geo (nrows, ncols) – subplot layout of the figure. subidx (int): index of the axis obj in the (nrows, ncols) layout.

i.e. 1 for the 1st subplot.

fontsize (int): default font size. This is determined from an
empirical formula that scales down the default font size for a bigger grid layout.

`getGrid`()

Get x- and y- coordnates

Returns

xarray (1darray) – 1d array of the x-coordinates. yarray (1darray): 1d array of the y-coordinates. lons,lats (ndarray): 2d array of the x- and y- coordinates, as

created from *lons, lats = np.meshgrid(xarray, yarray)*.

`getLabelBool`()

Decide whether to plot axis ticks and ticklabels on the 4 sides.

Returns

parallels (list) –

boolean flag for the x-axis ticks/labels on 4 sides:
[left, right, top, bottom]

meridians (list): boolean flag for the y-axis ticks/labels on 4 sides:
[left, right, top, bottom]

getLabelBoolForShareXY(*geo*, *idx*)

Decide ticks and ticklabels on the 4 sides with shared x and y.

Parameters

- **geo** (*nrows, ncols*) – subplot layout of the figure.
- **idx** (*int*) – index of the axis obj in the (nrows, ncols) layout. i.e. 1 for the 1st subplot.

Returns

parallels (list) –

boolean flag for the x-axis ticks/labels on 4 sides:
[left, right, top, bottom]

meridians (list): boolean flag for the y-axis ticks/labels on 4 sides:
[left, right, top, bottom]

plot()

Main plotting interface

Calls the core plotting function self._plot(), which handles the 2D plotting depending on the plotting method. Then plots axes, colorbar and title.

Returns

self.cs (mappable) –

the mappable obj, e.g. return value from contour()
or contourf().

plotAxes()

Plot axes ticks and ticklabels

plotColorbar()

Plot colorbar

Returns
cbar (matplotlib colorbar obj) – colorbar obj.

Only creates a colorbar for isofill/contourf or isoline/contour plots.

plotTitle()

Plot title

Use self.title as the figure title if self.ax is the single plot in the figure. If None, automatically get an alphabetic subtitle if self.ax is a subplot, e.g. '(a)' for the 1st subplot, '(d)' for the 4th one. If self.title is str and self.ax is a subplot, prepend self.title with the alphabetic index. One can force overriding the alphabetic index by giving a title str in the format of '(x) xxxx', e.g. '(p) subplot-p'.

class base_utils.**Plot2Quiver**(*u*, *v*, *method*, *ax=None*, *xarray=None*, *yarray=None*, *title=None*, *label_axes=True*, *axes_grid=False*, *clean=False*, *fontsize=None*, *units=None*, *fill_color='w'*, *curve=False*)

2D vector plotting class

For geographical vector plots, see Plot2QuiverBasemap or Plot2QuiverCartopy, which handles equivalent plotting with geographical map projections.

__init__(*u*, *v*, *method*, *ax=None*, *xarray=None*, *yarray=None*, *title=None*, *label_axes=True*, *axes_grid=False*, *clean=False*, *fontsize=None*, *units=None*, *fill_color='w'*, *curve=False*)

Parameters

- **u** (*ndarray*) – x- and y-component of velocity to plot. Mush have dimensions >= 2. For data with rank>2, take the slab from the last 2 dimensions.
- **v** (*ndarray*) – x- and y-component of velocity to plot. Mush have dimensions >= 2. For data with rank>2, take the slab from the last 2 dimensions.
- **method** (*Quiver obj*) – quiver plotting method. Determines how to plot the quivers.

Keyword Arguments

- **ax** (*matplotlib axis or None*) – axis obj. Determines where to plot. If None, create a new.
- **xarray** (*1darray or None*) – array to use as the x-coordinates. If None, use the indices of the last dimension: np.arange(slab.shape[-1]).
- **yarray** (*1darray or None*) – array to use as the y-coordinates. If None, use the indices of the 2nd last dimension: np.arange(slab.shape[-2]).
- **title** (*str or None*) – text as the figure title if <ax> is the single plot in the figure. If None, automatically get an alphabetic subtitle if <ax> is a subplot, e.g. '(a)' for the 1st subplot, '(d)' for the 4th one. If str and <ax> is a subplot, prepend <title> with the alphabetic index. One can force overriding the alphabetic index by giving a title str in the format of '(x) xxxx', e.g. '(p) subplot-p'.
- **label_axes** (*bool or 'all' or tuple*) – controls axis ticks and ticklabels. If True, don't exert any inference other than changing the ticklabel fontsize, and let matplotlib put the ticks and ticklabels (i.e. default only left and bottom axes). If False, turn off all ticks and ticklabels. If 'all', plot ticks and ticks labels on all 4 sides. If (left, right, top, bottom), specify which side to plot ticks/ticklabels. Each swith is a bool or binary. If None, will set the ticks/ticklabels such that the interior subplots have no ticks/ticklabels, edge subplots have ticks/ticklabels on the outer edges, i.e. similar as the 'sharex', 'sharey' options. Location of the subplot is determined from return of *ax.get_geometry()*.
- **axes_grid** (*bool*) – whether to add axis grid lines.
- **clean** (*bool*) – if False, don't plot axis ticks/ticklabels, colorbar, axis grid lines or title.
- **fontsize** (*int*) – font size for ticklabels, title, axis labels, colorbar ticklabels.
- **units** (*str or None*) – unit of <u> and <v>. Will be plotted next to the reference vector.
- **fill_color** (*str or color tuple*) – color to use as background color. If data have missings, they will be shown as this color.
- **curve** (*bool*) – whether to plot quivers as curved vectors. Experimental.

plot()

Main plotting interface

Calls the core plotting function self._plot(), which handles the 2D plotting using quiver plotting method. Then plots axes, quiverkey and title.

Returns

self.quiver (mappable) – the quiver obj, i.e. return value quiver().

plotkey()

Plot the reference quiver key

Returns

quiverkey (quiver key).

class base_utils.**Quiver**(*step=1*, *reso=None*, *scale=None*, *keylength=None*, *linewidth=0.0015*, *color='k'*, *alpha=1.0*)

Plotting method for quiver plots

__init__(*step=1*, *reso=None*, *scale=None*, *keylength=None*, *linewidth=0.0015*, *color='k'*, *alpha=1.0*)

Plotting method for quiver plots

Keyword Arguments

- **step** (*int*) – sub-sample steps in both x- and y- axes. U and V data are sub-sampled using *U[::step,::step]*.
- **reso** (*int or None*) – if not None, regrid input U and V data to a lower resolution, measured in grids. If both < reso > and <step> are given, use <reso>. Requires scipy for this functionality.
- **scale** (*float or None*) – see same arg as matplotlib.pyplot.quiver().
- **keylength** (*float or None*) – see same arg as matplotlib.pylot.quiver().
- **linewidth** (*float*) – line width.
- **color** (*str or color tuple*) – color to plot quiver arrows.
- **alpha** (*float*) – transparent level in [0, 1].

class base_utils.**Shading**(*color='0.5'*, *alpha=0.5*)

Plotting method for shading plots

__init__(*color='0.5'*, *alpha=0.5*)

Plotting method for shading plots

Keyword Arguments

- **color** (*str or color tuple*) – color of shading.
- **alpha** (*float*) – transparent level, in range of [0, 1].

base_utils.**alternateTicks**(*cbar*, *ticks=None*, *fontsize=9*)

Create alternating ticks and ticklabels for colorbar

Parameters
cbar (*matplotlib colorbar obj*) – input colorbar obj to alter.

Keyword Arguments

- **ticks** (*list or array or None*) – ticks of the colorbar. If None, get from cbar.get_ticks().
- **fontsize** (*str*) – font size for tick labels.

Returns
cbar (matplotlib colorbar obj) – the altered colorbar.

Only works for horizontal colorbar with discrete ticks. As vertical colorbar doesn't tend to have overlapping tick labels issue.

base_utils.**getColorbarPad**(*ax*, *orientation*, *base_pad=0.0*)

Compute padding value for colorbar axis creation

Parameters

- **ax** (*Axis obj*) – axis object used as the parent axis to create colorbar.
- **orientation** (*str*) – 'horizontal' or 'vertical'.

Keyword Arguments
base_pad (*float*) – default pad. The resultant pad value is the computed space + base_pad.

Returns
pad (float) – the pad argument passed to make_axes_gridspec() function.

base_utils.**getColormap**(*cmap*)

Get a colormap

Parameters
cmap (*matplotlib colormap, str or None*) – if colormap, return as is. if str, get a colormap by name: getattr(plt.cm, cmap). If None, use default of rcParams['default_cmap'].

Returns
cmap (matplotlib colormap) – matplotlib colormap.

base_utils.**getMissingMask**(*slab*)

Get a bindary array denoting missing (masked or nan).

Parameters
slab (*ndarray*) – input array that may contain masked values or nans.

Returns

mask (ndarray) –

bindary array with same shape as <slab> with 1s for
missing, 0s otherwise.

base_utils.**getQuantiles**(*slab*, *quantiles=None*, *verbose=True*)

Find quantiles of a slab

Parameters
slab (*ndarray*) – input ndarray whose quantiles will be found.

Keyword Arguments
quantiles (*float or a list of floats*) – desired quantiles(s).

Returns
results (ndarray) – 1darray, left quantiles

base_utils.**getRange**(*vars*, *min_level=None*, *max_level=None*, *ql=None*, *qr=None*, *verbose=True*)

Get min/max value

Parameters
vars (*list*) – a list of ndarrays.

Keyword Arguments

- **min_level** (*None or float*) – given minimum level.
- **max_level** (*None or float*) – given maximum level.
- **ql** (*None or float*) – given left quantile.
- **qr** (*None or float*) – given right quantile.

Returns
vmin (float) – lowest level to take from variables. vmax (float): highest level to take from variables. data_min (float): lowest level among variables. data_max (float): highest level among variables.

base_utils.**getSlab**(*var*, *index1=-1*, *index2=-2*, *verbose=True*)

Get a slab from a variable

Parameters
var – (ndarray): ndarray with dimension >=2.

Keyword Arguments
index1,index2 (*int*) – indices denoting the dimensions that define a 2d slab.

Returns

slab (ndarray) –

the (1st) slab from <var>.
E.g. <var> has dimension (12,1,241,480), getSlab(var) will return the 1st time point with singleton dimension squeezed.

base_utils.**index2Letter**(*index*, *verbose=True*)

Translate an integer index to letter index

Parameters
index (*int*) – integer index for a subplot.

Returns

letter (str) – corresponding letter index for <index>.

1 (a) 2 (b) 3 (c) … … 27 (aa) … 52 (zz)

base_utils.**mkscale**(*n1*, *n2*, *nc=12*, *zero=1*)

Create nice looking levels given a min and max.

Parameters

- **n1** (*floats*) – min and max levels between which to create levels.
- **n2** (*floats*) – min and max levels between which to create levels.

Keyword Arguments

- **nc** (*int*) – suggested number of levels. Note that the resulant levels may not have the exact number of levels as required.
- **zero** (*int*) – Not all implemented yet so set to 1 but values will be: -1: zero MUST NOT be a contour

  0: let the function decide # NOT IMPLEMENTED 1: zero CAN be a contour (default)
  2: zero MUST be a contour

Returns

cnt (list) –

a list of levels between approximately <n1> and <n2>,
with a number of levels more or less as <nc>.

Examples of Use: >>> vcs.mkscale(0,100) [0.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0] >>> vcs.mkscale(0,100,nc=5) [0.0, 20.0, 40.0, 60.0, 80.0, 100.0] >>> vcs.mkscale(-10,100,nc=5) [-25.0, 0.0, 25.0, 50.0, 75.0, 100.0] >>> vcs.mkscale(-10,100,nc=5,zero=-1) [-20.0, 20.0, 60.0, 100.0] >>> vcs.mkscale(2,20) [2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0] >>> vcs.mkscale(2,20,zero=2) [0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0]

Copied from vcs/util.py

base_utils.**pickPoint**(*ax*, *color='y'*)

Pick points from plot and store coordinates.

Parameters
ax (*matplotlib axis*) – axis from whose plot to pick points.

Keyword Arguments
color (*str or RGB tuple*) – Color of picked points.

Returns
points (list) – list of (x,y) coordinates.

base_utils.**plot2**(*var*, *method*, *ax=None*, *xarray=None*, *yarray=None*, *var_v=None*, ***kwargs*)

Wrapper 2D plotting interface function

Parameters

- **var** (*ndarray*) – input data to plot. Determines what to plot. Mush have dimensions >= 2. For data with rank>2, take the slab from the last 2 dimensions.
- **method** (*PlotMethod*) – plotting method. Determines how to plot. Could be Isofill, Isoline, Boxfill, Quiver, Shading, Hatch, GIS.

Keyword Arguments

- **ax** (*matplotlib axis or None*) – axis obj. Determines where to plot. If None, create a new.
- **xarray** (*1darray or None*) – array to use as the x-coordinates. If None, use the indices of the last dimension: np.arange(slab.shape[-1]).
- **yarray** (*1darray or None*) – array to use as the y-coordinates. If None, use the indices of the 2nd last dimension: np.arange(slab.shape[-2]).
- **var_v** (*ndarray or None*) – if a quiver plot (method is Quiver), the y-component of the velocity data, and <var> is the x-component.
- **nc_interface** (*str*) – netcdf data interfacing module, could be 'cdat', 'xarray', 'iris' or 'netcdf4'.
- **geo_interface** (*str*) – geographical plotting module, could be 'basemap', or 'cartopy'.
- **isgeomap** (*bool*) – whether to use geographcial plot.
- **projection** (*str*) – if use geographical plot, the map projection.
- **bmap** (*basemap obj or None*) – reuse an existing basemap obj if not None.
- **title** (*str or None*) – text as the figure title if <ax> is the single plot in the figure. If None, automatically get an alphabetic subtitle if <ax> is a subplot, e.g. '(a)' for the 1st subplot, '(d)' for the 4th one. If str and <ax> is a subplot, prepend <title> with the alphabetic index. One can force overriding the alphabetic index by giving a title str in the format of '(x) xxxx', e.g. '(p) subplot-p'.
- **label_axes** (*bool or 'all' or tuple*) – controls axis ticks and ticklabels. If True, don't exert any inference other than changing the ticklabel fontsize, and let matplotlib put the ticks and ticklabels (i.e. default only left and bottom axes). If False, turn off all ticks and ticklabels. If 'all', plot ticks and ticks labels on all 4 sides. If (left, right, top, bottom), specify which side to plot ticks/ticklabels. Each swith is a bool or binary. If None, will set the ticks/ticklabels such that the interior subplots have no ticks/ticklabels, edge subplots have ticks/ticklabels on the outer edges, i.e. similar as the 'sharex', 'sharey' options. Location of the subplot is determined from return of *ax.get_geometry()*.

- **axes_grid** (*bool*) – whether to add axis grid lines.
- **legend** (*str or None*) – controls whether to share colorbar or not. A colorbar is only plotted for Isofill/Isoline plots. If None, don't put colorbar. If 'local', <ax> has its own colorbar. If 'global', all subplots in the figure share a single colorbar, which is created by the 1st subplot in the figure, which is determined from the return of *ax.get_geometry()*.
- **legend_ori** (*str*) – orientation of colorbar. 'horizontal' or 'vertical'.
- **clean** (*bool*) – if False, don't plot axis ticks/ticklabels, colorbar, axis grid lines or title.
- **fontsize** (*int*) – font size for ticklabels, title, axis labels, colorbar ticklabels.
- **fix_aspect** (*bool*) – passed to the constructor of basemap: *Basemap(xxx, fix_aspect=fix_aspect)*.
- **fill_color** (*str or color tuple*) – color to use as background color. If data have missings, they will be shown as this color. It is better to use a grey than while to better distinguish missings.

Returns
plotobj (Plot2D obj).

base_utils.**regridToReso**(*var*, *inlat*, *inlon*, *dlat*, *dlon*, *lat_idx=-2*, *lon_idx=-1*, *method='linear'*, *return_coords=False*, *verbose=True*)

Regrid to given resolution, using scipy

Parameters

- **var** (*ndarray*) – input nd array.
- **inlat** (*1darray*) – input latitude coordinates.
- **inlon** (*1darray*) – input longitude coordinates.
- **dlat** (*float*) – target latitudinal resolution.
- **dlon** (*float*) – target longitudinal resolution.

Keyword Arguments

- **lat_idx** (*int*) – index for the latitude dimension.
- **lon_idx** (*int*) – index for the longitude dimension.
- **method** (*str*) – interpolation method, could be 'linear' or 'nearest'.
- **return_coords** (*bool*) – if True, also return new lat/lon coordinates.

Returns
result (ndarray) – interpolated result. newlat (1darray): if <return_coords> is True, the new latitude coordinates. newlon (1darray): if <return_coords> is True, the new longitude coordinates.

base_utils.**remappedColorMap2**(*cmap*, *vmin*, *vmax*, *vcenter*, *name='shiftedcmap'*)

Re-map the colormap to split positives and negatives.

Parameters

- **cmap** (*colormap*) – the matplotlib colormap to be altered.
- **vmin** (*float*) – minimal level in data.
- **vmax** (*float*) – maximal level in data.
- **vcenter** (*float*) – central level in data.

Keyword Arguments
name (*str*) – name for the altered colormap.

Returns

newcmap (colormap) –

re-mapped colormap such that:

if vmin < vmax <= vcenter:
0 in color map corresponds to vmin 0.5 in color map corresponds to vmax

if vcenter <= vmin < vmax:
0.5 in color map corresponds to vmin 1.0 in color map corresponds to vmax

E.g. if vcenter=0, this splits a diverging colormap to use only the negative/positive half the original colors.

base_utils.**restoreParams**()

Restore default parameters

6.2 Documentation page for basemap_utils.py

Basemap 2D plotting functions and classes.

Author: guangzhi XU (xugzhi1987@gmail.com) Update time: 2021-02-14 13:42:31.

class basemap_utils.**Plot2Basemap**(**args: Any*, ***kwargs: Any*)

2D geographical plotting class, using basemap

__init__(*var*, *method*, *xarray*, *yarray*, *ax=None*, *title=None*, *label_axes=True*, *axes_grid=False*, *legend=None*, *legend_ori=None*, *clean=False*, *fontsize=None*, *projection=None*, *fill_color=None*, *fix_aspect=False*, *isdrawcoastlines=True*, *isdrawcountries=True*, *isdrawcontinents=False*, *isdrawrivers=False*, *isfillcontinents=False*, *bmap=None*)

2D geographical plotting class, using basemap

Parameters

- **var** (*ndarray*) – input data to plot. Determines what to plot. Mush have dimensions >= 2. For data with rank>2, take the slab from the last 2 dimensions.
- **method** (*PlotMethod*) – plotting method. Determines how to plot. Could be Isofill, Isoline, Boxfill, Quiver, Shading, Hatch, GIS.
- **xarray** (*1darray or None*) – array to use as the x-coordinates. If None, use the indices of the last dimension: np.arange(slab.shape[-1]).
- **yarray** (*1darray or None*) – array to use as the y-coordinates. If None, use the indices of the 2nd last dimension: np.arange(slab.shape[-2]).

Keyword Arguments

- **ax** (*matplotlib axis or None*) – axis obj. Determines where to plot. If None, create a new.
- **title** (*str or None*) – text as the figure title if <ax> is the single plot in the figure. If None, automatically get an alphabetic subtitle if <ax> is a subplot, e.g. '(a)' for the 1st subplot, '(d)' for the 4th one. If str and <ax> is a subplot, prepend <title> with the alphabetic index. One can force overriding the alphabetic index by giving a title str in the format of '(x) xxxx', e.g. '(p) subplot-p'.

- **label_axes** (*bool or 'all' or tuple*) – controls axis ticks and ticklabels. If True, don't exert any inference other than changing the ticklabel fontsize, and let matplotlib put the ticks and ticklabels (i.e. default only left and bottom axes). If False, turn off all ticks and ticklabels. If 'all', plot ticks and ticks labels on all 4 sides. If (left, right, top, bottom), specify which side to plot ticks/ticklabels. Each swith is a bool or binary. If None, will set the ticks/ticklabels such that the interior subplots have no ticks/ticklabels, edge subplots have ticks/ticklabels on the outer edges, i.e. similar as the 'sharex', 'sharey' options. Location of the subplot is determined from return of *ax.get_geometry()*.
- **axes_grid** (*bool*) – whether to add axis grid lines.
- **legend** (*str or None*) – controls whether to share colorbar or not. A colorbar is only plotted for Isofill/Isoline plots. If None, don't put colorbar. If 'local', <ax> has its own colorbar. If 'global', all subplots in the figure share a single colorbar, which is created by the 1st subplot in the figure, which is determined from the return of *ax.get_geometry()*.
- **legend_ori** (*str*) – orientation of colorbar. 'horizontal' or 'vertical'.
- **clean** (*bool*) – if False, don't plot axis ticks/ticklabels, colorbar, axis grid lines or title.
- **fontsize** (*int*) – font size for ticklabels, title, axis labels, colorbar ticklabels.
- **projection** (*str*) – the map projection.
- **fill_color** (*str or color tuple*) – color to use as background color. If data have missings, they will be shown as this color. It is better to use a grey than while to better distinguish missings.
- **fix_aspect** (*bool*) – passed to the constructor of basemap: *Basemap(xxx, fix_aspect=fix_aspect)*.
- **isdrawcoastlines** (*bool*) – whether to draw continent outlines or not.
- **isdrawcountries** (*bool*) – whether to draw contry boundaries or not.
- **isdrawrivers** (*bool*) – whether to draw rivers or not.
- **isfillcontinents** (*bool*) – whether to fill continents or not.
- **bmap** (*basemap obj or None*) – reuse an existing basemap obj if not None.

createBmap()

Create basemap based on data domain

plotAxes()

Plot longitude/latitude ticks and ticklabels

Overwrites parent classes method

plotOthers()

Plot other map information

Plot continents, contries, rivers if needed.

class basemap_utils.**Plot2QuiverBasemap**(**args: Any*, ***kwargs: Any*)

2D geographical quiver plotting class, using basemap

__init__(*u*, *v*, *method*, *xarray*, *yarray*, *ax=None*, *title=None*, *label_axes=True*, *axes_grid=False*, *clean=False*, *fontsize=None*, *projection=None*, *units=None*, *fill_color='w'*, *curve=False*, *fix_aspect=False*, *isdrawcoastlines=True*, *isdrawcountries=True*, *isdrawcontinents=False*, *isdrawrivers=False*, *isfillcontinents=False*, *bmap=None*)

2D geographical quiver plotting class, using basemap

Parameters

- **u** (*ndarray*) – x- and y-component of velocity to plot. Mush have dimensions >= 2. For data with rank>2, take the slab from the last 2 dimensions.
- **v** (*ndarray*) – x- and y-component of velocity to plot. Mush have dimensions >= 2. For data with rank>2, take the slab from the last 2 dimensions.
- **method** (*Quiver obj*) – quiver plotting method. Determines how to plot the quivers.
- **xarray** (*1darray or None*) – array to use as the x-coordinates. If None, use the indices of the last dimension: np.arange(slab.shape[-1]).
- **yarray** (*1darray or None*) – array to use as the y-coordinates. If None, use the indices of the 2nd last dimension: np.arange(slab.shape[-2]).

Keyword Arguments

- **ax** (*matplotlib axis or None*) – axis obj. Determines where to plot. If None, create a new.
- **title** (*str or None*) – text as the figure title if <ax> is the single plot in the figure. If None, automatically get an alphabetic subtitle if <ax> is a subplot, e.g. '(a)' for the 1st subplot, '(d)' for the 4th one. If str and <ax> is a subplot, prepend <title> with the alphabetic index. One can force overriding the alphabetic index by giving a title str in the format of '(x) xxxx', e.g. '(p) subplot-p'.
- **label_axes** (*bool or 'all' or ((left_y, right_y, top_y, top_y)* – (left_x, right_x, top_x, top_x)) or None): controls axis ticks and ticklabels. If True, don't exert any inference other than changing the ticklabel fontsize, and let matplotlib put the ticks and ticklabels (i.e. default only left and bottom axes). If False, turn off all ticks and ticklabels. If 'all', plot ticks and ticks labels on all 4 sides. If ((left_y, right_y, top_y, top_y), (left_x, right_x, top_x, top_x)), specify which side to plot ticks/ticklabels. Each swith is a bool or binary. If None, will set the ticks/ticklabels such that the interior subplots have no ticks/ticklabels, edge subplots have ticks/ticklabels on the outer edges, i.e. similar as the 'sharex', 'sharey' options. Location of the subplot is determined from return of *ax.get_geometry()*.
- **axes_grid** (*bool*) – whether to add axis grid lines.
- **clean** (*bool*) – if False, don't plot axis ticks/ticklabels, colorbar, axis grid lines or title.
- **fontsize** (*int*) – font size for ticklabels, title, axis labels, colorbar ticklabels.
- **projection** (*str*) – the map projection.
- **units** (*str or None*) – unit of <u> and <v>. Will be plotted next to the reference vector.
- **fill_color** (*str or color tuple*) – color to use as background color. If data have missings, they will be shown as this color. It is better to use a grey than while to better distinguish missings.
- **curve** (*bool*) – whether to plot quivers as curved vectors. Experimental.
- **fix_aspect** (*bool*) – passed to the constructor of basemap: *Basemap(xxx, fix_aspect=fix_aspect)*.
- **isdrawcoastlines** (*bool*) – whether to draw continent outlines or not.
- **isdrawcountries** (*bool*) – whether to draw contry boundaries or not.
- **isdrawrivers** (*bool*) – whether to draw rivers or not.
- **isfillcontinents** (*bool*) – whether to fill continents or not.

- **bmap** (*basemap obj or None*) – reuse an existing basemap obj if not None.

plot()

Main plotting interface

Calls the core plotting function self._plot(), which handles the 2D plotting using quiver plotting method. Then plots axes, quiverkey and title.

Returns

self.quiver (mappable) – the quiver obj, i.e. return value quiver().

basemap_utils.**blueMarble**(*lat1*, *lon1*, *lat2*, *lon2*, *fig=None*, *projection='merc'*)

Plot bluemarble plot as background.

Parameters

- **lat1** (*floats*) – low-left corner. Longitude range 0-360
- **lon1** (*floats*) – low-left corner. Longitude range 0-360
- **lat2** (*floats*) – upper-right corner.
- **lon2** (*floats*) – upper-right corner.

Keyword Arguments

- **fig** (*matplotlib figure or None*) – If None, create a new.
- **projection** (*str*) – map projection.

NOTE: due to a bug in basemap, if the plot range is crossing the dateline, need to plot 2 separate plots joining at the dateline.

6.3 Documentation page for cdat_utils.py

Interfacing netcdf data via CDAT

Author: guangzhi XU (xugzhi1987@gmail.com) Update time: 2020-12-05 10:28:38.

cdat_utils.**checkGeomap**(*var*, *xarray*, *yarray*)

Check input args suitable for geo plot or not and do some preprocessing

Parameters

- **var** (*TransientVariable*) – input N-d TransientVariable.
- **xarray** (*ndarray*) – 1d array, x-coordinates.
- **yarray** (*ndarray*) – 1d array, y-coordinates.

Returns

isgeo (bool) –

True if inputs are suitable for geographical plot, False
otherwise.

var (TransientVariable): input <var> with latitude order reversed if
needed.

xx (ndarray): 1d array, use longitude axis of <var> if possible,
<xarray> otherwise

yy (ndarray): 1d array, use latitude axis of <var> if possible,
<yarray> otherwise

cdat_utils.**increasingLatitude**(*slab*, *verbose=False*)

Changes a slab so that is always has latitude running from south to north.

Parameters
slab (*TransientVariable*) – input TransientVariable, need to have a proper latitude axis.

Returns

slab2 (TransientVariable) –

if latitude axis is reversed, or <slab>
otherwise.

If <slab> has a latitude axis, and the latitudes run from north to south, a copy <slab2> is made with the latitudes reversed, i.e., running from south to north.

cdat_utils.**interpretAxis**(*axis*, *ref_var*, *verbose=True*)

Interpret and convert an axis id to index

Parameters

- **axis** (*int or str*) – axis option, integer (e.g. 0 for 1st dimension) or string (e.g. 'x' for x-dimension).
- **ref_var** (*TransientVariable*) – reference variable.

Returns
axis_index (int) – the index of required axis in <ref_var>.

E.g. index=interpretAxis('time',ref_var)
index=0

index=interpretAxis(1,ref_var) index=1

cdat_utils.**isInteger**(*x*)

Check an input is integer

Parameters
x (*unknow type*) – input

Returns
True if <x> is integer type, False otherwise.

cdat_utils.**readData**(*varid*)

Read sample netcdf data

Parameters
varid (*str*) – id of variable to read.

Returns
var (TransientVariable) – sample netcdf data.

6.4 Documentation page for netcdf4_utils.py

Interfacing netcdf data via netcdf4

Author: guangzhi XU (xugzhi1987@gmail.com; guangzhi.xu@outlook.com) Update time: 2021-01-24 17:30:56.

netcdf4_utils.**checkGeomap**(*var*, *xarray*, *yarray*)

Check input args suitable for geo plot or not and do some preprocessing

Parameters

- **var** (*TransientVariable*) – input N-d TransientVariable.
- **xarray** (*ndarray*) – 1d array, x-coordinates.
- **yarray** (*ndarray*) – 1d array, y-coordinates.

Returns

isgeo (bool) –

True if inputs are suitable for geographical plot, False otherwise.

var (TransientVariable): input <var> with latitude order reversed if needed.

xx (ndarray): 1d array, use longitude axis of <var> if possible, <xarray> otherwise

yy (ndarray): 1d array, use latitude axis of <var> if possible, <yarray> otherwise

netcdf4_utils.**readData**(*varid*)

Read in a variable from an netcdf file

Parameters

- **abpath_in** (*str*) – absolute file path to the netcdf file.
- **varid** (*str*) – id of variable to read.

Returns

ncvarNV (NCVAR) – variable stored as an NCVAR obj.

6.5 Documentation page for cartopy_utils.py

Cartopy related utilities

Author: guangzhi XU (xugzhi1987@gmail.com) Update time: 2020-12-05 10:28:38.

class cartopy_utils.**Plot2Cartopy**(**args: Any*, ***kwargs: Any*)

class cartopy_utils.**Plot2QuiverCartopy**(**args: Any*, ***kwargs: Any*)

CHAPTER

SEVEN

GITHUB AND CONTACT

The code of this package is hosted at https://github.com/Xunius/gplot.

For any queries, please contact xugzhi1987@gmail.com.

CHAPTER

EIGHT

CONTRIBUTING AND GETTING HELP

We welcome contributions from the community. Please create a fork of the project on GitHub and use a pull request to propose your changes. We strongly encourage creating an issue before starting to work on major changes, to discuss these changes first.

For help using the package, please post issues on the project GitHub page.

CHAPTER

NINE

LICENSE

license

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

G

H

I

M

N

P

Q

R

S