GooseMPL Documentation

Tom de Geus

Apr 25, 2019

Contents

GooseMPL provides a style and several style extensions for matplotlib, some custom functions that extend matplotlib, and several examples to make professional plots using matplotlib.

To obtain GooseMPL one can do either of the following.

1. Using PyPi:

   ```
   $ pip install GooseMPL
   ```

   Then install the matplotlib-style files using:

   ```
   >>> import GooseMPL
   >>> GooseMPL.copy_style()
   ```

   You'll only have to do this once after installing/updating.

2. Manual installation. After downloading run

   ```
   $ python setup.py install
   ```

   The matplotlib-style files are automatically copied during this process.

Note: This library is free to use under the MIT license. Any additions are very much appreciated, in terms of suggested functionality, code, documentation, testimonials, word of mouth advertisement, ... Bug reports or feature requests can be filed on GitHub. As always, the code comes with no guarantee. None of the developers can be held responsible for possible mistakes.

Download: .zip file | .tar.gz file.

(c - MIT) T.W.J. de Geus (Tom) | tom@geus.me | www.geus.me | github.com/tdegeus/GooseMPL

CHAPTER 1

Contents

1.1 Customized style

1.1.1 Styles in GooseMPL

The following styles are part of GooseMPL (they become available upon installing GooseMPL):

- `goose`

  Customized layout settings.

- `goose-latex`

  Extend a style to enable the use of LaTeX, and change the font to LaTeX default Computer Modern font.

- `goose-tick-in`

  Place the tick-markers (*the little lines*) at the inside of the axes rather than at the outside.

- `goose-tick-lower`

  Shown only axes on the bottom and left side of the figure (those on the top and right are not shown).

See the *Examples*.

1.1.2 Background

Matplotlib has a very convenient way to customize plots while minimizing the amount of customized code needed for this. It employs easy-to-switch plotting styles with the same parameters as a `matplotlibrc` file. The only thing needed to switch styles is:

```
import matplotlib.pyplot as plt
plt.style.use('name_of_custom_style')
```

A number of styles are available. To list them use `plt.style.available`.

Also, one can use one's own style. This is a plain-text file `name_of_custom_style.mplstyle` stored in a sub-directory `stylelib` of the Matplotlib configuration directory; e.g.:

```
~/.matplotlib/stylelib/          # MacOS/Linux
~/.config/matplotlib/stylelib/   # MacOS/Linux
```

The exact directory depends on the operating system and the installation. To find the directory to use on your system, use:

```
import matplotlib
matplotlib.get_configdir()
```

Note: More information in the matplotlib documentation

1.1.3 Tips

Combining styles

Combining different styles is easily accomplished by including a list of styles. For example:

```
plt.style.use(['dark_background','presentation'])
```

Temporary styling

To compose parts of the plot with a different style use:

```
with plt.style.context(('presentation')):
    plt.plot(np.sin(np.linspace(0, 2 * np.pi)))
```

Extending

To get the available fields do the following:

```
import matplotlib as mpl

print(mpl.rcParams)
```

1.2 Custom plot functions

1.2.1 Overview

Set axes limits

GooseMPL.scale_lim(lim[, factor])	Scale limits to be 5% wider, to have a nice plot.
GooseMPL.set_decade_lims([axis, direction])	Set limits the the floor/ceil values in terms of decades.

Plot in relative coordinates

GooseMPL.plot(x, y[, units, axis])	Plot.
GooseMPL.text(x, y, text[, units, axis])	Plot a text.
GooseMPL.rel2abs_x(x[, axis])	Transform relative x-coordinates to absolute x-coordinates.
GooseMPL.rel2abs_y(y[, axis])	Transform relative y-coordinates to absolute y-coordinates.
GooseMPL.abs2rel_x(x[, axis])	Transform absolute x-coordinates to relative x-coordinates.
GooseMPL.abs2rel_y(y[, axis])	Transform absolute y-coordinates to relative y-coordinates.

Annotate power-law

GooseMPL.grid_powerlaw(exp[, insert, skip, ...])	Draw a power-law grid: a grid that respects a certain power-law exponent.
GooseMPL.plot_powerlaw(exp, startx, starty)	Plot a power-law.
GooseMPL.annotate_powerlaw(text, exp, ...[, ...])	Added a label to the middle of a power-law annotation (see `goosempl.plot_powerlaw`).
GooseMPL.diagonal_powerlaw(exp[, ll, lr, ...])	Set the limits such that a power-law with a certain exponent lies on the diagonal.

Plot mesh

GooseMPL.patch(*args, **kwargs)	Add patches to plot.

(Plot) statistics

GooseMPL.histogram(data[, return_edges])	Compute histogram.
GooseMPL.histogram_bin_edges(data[, bins, ...])	Determine bin-edges.
GooseMPL.histogram_bin_edges_minwidth(...)	Merge bins with right-neighbour until each bin has a minimum width.
GooseMPL.histogram_bin_edges_mincount(data, ...)	Merge bins with right-neighbour until each bin has a minimum number of data-points.
GooseMPL.histogram_cumulative(data, **kwargs)	Compute cumulative histogram.
GooseMPL.cdf(data[, mode])	Return cumulative density.
GooseMPL.hist(P, edges, **kwargs)	Plot histogram.

Styles

GooseMPL.subplots([scale_x, scale_y, scale])	Run `matplotlib.pyplot.subplots` with `figsize` set to the correct multiple of the default.
GooseMPL.copy_style()	Write all goose-styles to the relevant matplotlib configuration directory.

Continued on next page

Table 6 – continued from previous page

GooseMPL.find_latex_font_serif()	Find an available font to mimic LaTeX, and return its name.

1.2.2 Documentation

This module provides some extensions to matplotlib.

dependencies

- numpy
- matplotlib

copyright

Tom de Geus
tom@geus.me
http://www.geus.me

GooseMPL.**abs2rel_x**(*x*, *axis=None*)

Transform absolute x-coordinates to relative x-coordinates. Relative coordinates correspond to a fraction of the relevant axis. Be sure to set the limits and scale before calling this function!

Arguments

x (float, list) Absolute coordinates.

Options

axis ([plt.gca()]|...) Specify the axis to which to apply the limits.

Returns

x (float, list) Relative coordinates.

GooseMPL.**abs2rel_y**(*y*, *axis=None*)

Transform absolute y-coordinates to relative y-coordinates. Relative coordinates correspond to a fraction of the relevant axis. Be sure to set the limits and scale before calling this function!

Arguments

y (float, list) Absolute coordinates.

Options

axis ([plt.gca()]|...) Specify the axis to which to apply the limits.

Returns

y (float, list) Relative coordinates.

GooseMPL.**annotate_powerlaw**(*text*, *exp*, *startx*, *starty*, *width=None*, *rx=0.5*, *ry=0.5*, ***kwargs*)

Added a label to the middle of a power-law annotation (see `goosempl.plot_powerlaw`).

Arguments

exp (float) The power-law exponent.

startx, starty (float) Start coordinates.

Options

width, height, endx, endy (float) Definition of the end coordinate (only on of these options is needed).

rx, ry (`float`) Shift in x- and y-direction w.r.t. the default coordinates.

units (`['relative']|'absolute'`) The type of units in which the coordinates are specified. Relative coordinates correspond to a fraction of the relevant axis. If you use relative coordinates, be sure to set the limits and scale before calling this function!

axis (`[plt.gca()]|...`) Specify the axis to which to apply the limits.

... Any `plt.text(...)` option.

Returns The handle of the `plt.text(...)` command.

`GooseMPL.`**`cdf`**(*data*, *mode='continuous'*, ***kwargs*)
Return cumulative density.

Arguments

data (`<numpy.ndarray>`) Input data, to plot the distribution for.

Returns

P (`<numpy.ndarray>`) Cumulative probability.

x (`<numpy.ndarray>`) Data points.

`GooseMPL.`**`copy_style`**()
Write all goose-styles to the relevant matplotlib configuration directory.

`GooseMPL.`**`diagonal_powerlaw`**(*exp*, *ll=None*, *lr=None*, *tl=None*, *tr=None*, *width=None*, *height=None*, *plot=False*, ***kwargs*)
Set the limits such that a power-law with a certain exponent lies on the diagonal.

Arguments

exp (`<float>`) The power-law exponent.

ll, lr, tl, tr (`<list>`) Coordinates of the lower-left, or the lower-right, or the top-left, or the top-right corner.

width, height (`<float>`) Width or the height.

Options

axis (`[plt.gca()]|...`) Specify the axis to which to apply the limits.

plot (`[False]|True`) Plot the diagonal.

... Any `plt.plot(...)` option.

Returns The handle of the `plt.plot(...)` command (if any).

`GooseMPL.`**`find_latex_font_serif`**()
Find an available font to mimic LaTeX, and return its name.

`GooseMPL.`**`grid_powerlaw`**(*exp*, *insert=0*, *skip=0*, *end=-1*, *step=0*, *axis=None*, ***kwargs*)
Draw a power-law grid: a grid that respects a certain power-law exponent. The grid-lines start from the positions of the ticks.

Arguments

exp (`float`) The power-law exponent.

Options

insert (`<int>`) Insert extra lines in between the default lines set by the tick positions.

skip, end, step (<int>) Select from the lines based on `coor = coor[skip:end:step]`.

axis ([plt.gca()]|...) Specify the axis to which to apply the limits.

... Any `plt.plot(...)` option.

Returns The handle of the `plt.plot(...)` command.

`GooseMPL.`**`hist`**(*P*, *edges*, ***kwargs*)
Plot histogram.

`GooseMPL.`**`histogram`**(*data*, *return_edges=True*, ***kwargs*)
Compute histogram. See numpy.histrogram

Extra options

return_edges ([True]|[False]) Return the bin edges if set to `True`, return their midpoints otherwise.

`GooseMPL.`**`histogram_bin_edges`**(*data*, *bins=10*, *mode='equal'*, *min_count=None*, *integer=False*, *remove_empty_edges=True*, *min_width=None*)
Determine bin-edges.

Arguments

data (<array_like>) Input data. The histogram is computed over the flattened array.

Options

bins ([10]|<int>) The number of bins.

mode (['equal'|<str>]) Mode with which to compute the bin-edges: * `'equal'`: each bin has equal width. * `'log'`: logarithmic spacing. * `'uniform'`: uniform number of data-points per bin.

min_count (<int>) The minimum number of data-points per bin.

min_width (<float>) The minimum width of each bin.

integer ([False]|[True]) If `True`, bins not encompassing an integer are removed (e.g. a bin with edges `[1.1, 1.9]` is removed, but `[0.9, 1.1]` is not removed).

remove_empty_edges ([True]|[False]) Remove empty bins at the beginning or the end.

Returns

bin_edges (<array of dtype float>) The edges to pass into histogram.

`GooseMPL.`**`histogram_bin_edges_mincount`**(*data*, *min_count*, *bins*)
Merge bins with right-neighbour until each bin has a minimum number of data-points.

Arguments

data (<array_like>) Input data. The histogram is computed over the flattened array.

bins (<array_like>|<int>) The bin-edges (or the number of bins, automatically converted to equal-sized bins).

min_count (<int>) The minimum number of data-points per bin.

`GooseMPL.`**`histogram_bin_edges_minwidth`**(*min_width*, *bins*)
Merge bins with right-neighbour until each bin has a minimum width.

Arguments

bins (<array_like>) The bin-edges.

min_width (<float>) The minimum bin width.

GooseMPL.**histogram_cumulative**(*data*, ***kwargs*)

Compute cumulative histogram. See numpy.histrogram

Extra options

return_edges ([True] | [False]) Return the bin edges if set to `True`, return their midpoints otherwise.

normalize ([False] | True) Normalize such that the final probability is one. In this case the function returns the (binned) cumulative probability density.

GooseMPL.**patch**(**args*, ***kwargs*)

Add patches to plot. The color of the patches is indexed according to a specified color-index.

Example Plot a finite element mesh: the outline of the undeformed configuration, and the deformed configuration for which the elements get a color e.g. based on stress:

```
import matplotlib.pyplot as plt
import goosempl          as gplt

fig,ax = plt.subplots()

p = gplt.patch(coor=coor+disp,conn=conn,axis=ax,cindex=stress,cmap=
↪'YlOrRd',edgecolor=None)
_ = gplt.patch(coor=coor     ,conn=conn,axis=ax)

cbar = fig.colorbar(p,axis=ax,aspect=10)

plt.show()
```

Arguments - option 1/2

patches (<list>) List with patch objects. Can be replaced by specifying `coor` and `conn`.

Arguments - option 2/2

coor (<numpy.ndarray> | <list> (nested)) Matrix with on each row the coordinates (positions) of each node.

conn (<numpy.ndarray> | <list> (nested)) Matrix with on each row the number numbers (rows in `coor`) which form an element (patch).

Options

cindex (<numpy.ndarray>) Array with, for each patch, the value that should be indexed to a color.

axis (<matplotlib>) Specify an axis to include to plot in. By default the current axis is used.

autoscale ([True] | False) Automatically update the limits of the plot (currently automatic limits of Collections are not supported by matplotlib).

Recommended options

cmap (<str> | ...) Specify a colormap.

linewidth (<float>) Width of the edges.

edgecolor (<str> | ...) Color of the edges.

clim ((<float>,<float>)) Lower and upper limit of the color-axis.

Returns

handle (<matplotlib>) Handle of the patch objects.

See also:

- matplotlib example.

GooseMPL.**plot**(*x*, *y*, *units='absolute'*, *axis=None*, ***kwargs*)

Plot.

Arguments

x, y (list) Coordinates.

Options

units (['absolute']|'relative') The type of units in which the coordinates are specified. Relative coordinates correspond to a fraction of the relevant axis. If you use relative coordinates, be sure to set the limits and scale before calling this function!

... Any `plt.plot(...)` option.

Returns The handle of the `plt.plot(...)` command.

GooseMPL.**plot_powerlaw**(*exp*, *startx*, *starty*, *width=None*, ***kwargs*)

Plot a power-law.

Arguments

exp (float) The power-law exponent.

startx, starty (float) Start coordinates.

Options

width, height, endx, endy (float) Definition of the end coordinate (only on of these options is needed).

units (['relative']|'absolute') The type of units in which the coordinates are specified. Relative coordinates correspond to a fraction of the relevant axis. If you use relative coordinates, be sure to set the limits and scale before calling this function!

axis ([plt.gca()]|...) Specify the axis to which to apply the limits.

... Any `plt.plot(...)` option.

Returns The handle of the `plt.plot(...)` command.

GooseMPL.**rel2abs_x**(*x*, *axis=None*)

Transform relative x-coordinates to absolute x-coordinates. Relative coordinates correspond to a fraction of the relevant axis. Be sure to set the limits and scale before calling this function!

Arguments

x (float, list) Relative coordinates.

Options

axis ([plt.gca()]|...) Specify the axis to which to apply the limits.

Returns

x (float, list) Absolute coordinates.

GooseMPL.**rel2abs_y**(*y*, *axis=None*)
Transform relative y-coordinates to absolute y-coordinates. Relative coordinates correspond to a fraction of the relevant axis. Be sure to set the limits and scale before calling this function!

Arguments

y (`float`, `list`) Relative coordinates.

Options

axis ([`plt.gca()`]|`...`) Specify the axis to which to apply the limits.

Returns

y (`float`, `list`) Absolute coordinates.

GooseMPL.**scale_lim**(*lim*, *factor=1.05*)
Scale limits to be 5% wider, to have a nice plot.

Arguments

lim (`<list>`|`<str>`) The limits. May be a string "[...,...]", which is converted to a list.

Options

factor ([`1.05`]|`<float>`) Scale factor.

GooseMPL.**set_decade_lims**(*axis=None*, *direction=None*)
Set limits the the floor/ceil values in terms of decades.

Options

axis ([`plt.gca()`]|`...`) Specify the axis to which to apply the limits.

direction ([`None`]|`'x'`|`'y'`) Limit the application to a certain direction (default: both).

GooseMPL.**subplots**(*scale_x=None*, *scale_y=None*, *scale=None*, ***kwargs*)
Run `matplotlib.pyplot.subplots` with `figsize` set to the correct multiple of the default.

Additional options

scale, scale_x, scale_y (`<float>`) Scale the figure-size (along one of the dimensions).

GooseMPL.**text**(*x*, *y*, *text*, *units='absolute'*, *axis=None*, ***kwargs*)
Plot a text.

Arguments

x, y (`float`) Coordinates.

text (`str`) Text to plot.

Options

units ([`'absolute'`]|`'relative'`) The type of units in which the coordinates are specified. Relative coordinates correspond to a fraction of the relevant axis. If you use relative coordinates, be sure to set the limits and scale before calling this function!

... Any `plt.text(...)` option.

Returns The handle of the `plt.text(...)` command.

1.3 Examples

Outline

1.3.1 Plot

Basic

[source: plot.py]

```
import numpy             as np
import matplotlib.pyplot as plt

plt.style.use(['goose','goose-latex'])
```

(continues on next page)

(continued from previous page)

```
x = np.linspace(0,2*np.pi,400)

fig,ax = plt.subplots()

ax.plot(x,np.sin(x)          ,label=r'$\sin \big( x \big)$')
ax.plot(x,np.sin(x-np.pi/4.),label=r'$\sin \big( x - \tfrac{\pi}{4} \big)$')

ax.set_title('Simple plot')

ax.xaxis.set_ticklabels(['0',r'$\pi$',r'$2\pi$'])
ax.xaxis.set_ticks([0,np.pi,2*np.pi])
ax.yaxis.set_ticks([-1,0,1])

ax.legend(loc='upper right')

ax.set_xlim([0,2*np.pi])

ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')

plt.savefig('plot.svg')
plt.show()
```

Subplot

[source: subplot.py]

```
import numpy             as np
import matplotlib.pyplot as plt

plt.style.use(['goose','goose-latex'])

# --- some data ---

x = np.linspace(0,2*np.pi,400)

# --- open figure with 2 subplots ---

fig, axes = plt.subplots(ncols=2, figsize=(18,6))

ax1 = axes[0]
ax2 = axes[1]

# --- first subplot ---

ax1.plot(x,np.sin(x)          ,label=r'$\sin \big( x \big)$')
ax1.plot(x,np.sin(x-np.pi/4.),label=r'$\sin \big( x - \tfrac{\pi}{4} \big)$')

ax1.set_title('First subplot')
```

(continues on next page)

(continued from previous page)

```
ax1.xaxis.set_ticklabels(['0',r'$\pi$',r'$2\pi$'])
ax1.xaxis.set_ticks([0,np.pi,2*np.pi])
ax1.yaxis.set_ticks([-1,0,1])

ax1.legend(loc='upper right')

ax1.set_xlim([0,2*np.pi])

ax1.set_xlabel(r'$x$')
ax1.set_ylabel(r'$y$')

# --- second subplot ---

ax2.plot(x,np.cos(x)          ,linestyle='--',label=r'$\cos \big( x \big)$')
ax2.plot(x,np.cos(x-np.pi/4.),linestyle='--',label=r'$\cos \big( x - \tfrac{\pi}{4}␣
↪\big)$')

ax2.set_title('Second subplot')

ax2.xaxis.set_ticklabels(['0',r'$\pi$',r'$2\pi$'])
ax2.xaxis.set_ticks([0,np.pi,2*np.pi])
ax2.yaxis.set_ticks([-1,0,1])

ax2.legend(loc='upper center')

ax2.set_xlim([0,2*np.pi])

ax2.set_xlabel(r'$x$')
ax2.set_ylabel(r'$y$')

# --- save/show ---

plt.savefig('subplot.svg')
plt.show()
```

Legend

Set background colour

[source: legend_background.py]

```
import numpy             as np
import matplotlib.pyplot as plt

plt.style.use(['goose', 'goose-latex'])

fig, ax = plt.subplots()

for i in range(10):
  ax.plot([0,1], [i/10,(i+10)/10], label=r'$i = {0:d}$'.format(i))
```

(continues on next page)

(continued from previous page)

```
ax.legend(loc='center right', facecolor='white', framealpha=1)

plt.savefig('legend_background.svg')
plt.show()
```

Move legend outside the plot

Note: References

StackOverflow - "How to put the legend out of matplotlib plot"

[source: legend_external.py]

```
import numpy             as np
import matplotlib.pyplot as plt

plt.style.use(['goose', 'goose-latex'])

fig, ax = plt.subplots(figsize=(9,6))

x = np.arange(10)

for i in range(5):
  ax.plot(x, i*x, label='$y = {:d}x$'.format(i))

legend = ax.legend(loc='center left', bbox_to_anchor=(1., 0.5), fancybox=True,␣
↪shadow=True)

frame = legend.get_frame()
frame.set_facecolor('white')
frame.set_edgecolor('black')

plt.savefig('legend_external.svg')
plt.show()
```

Line-color from colormap

[source: plot-cmap.py]

Note: References

StackOverflow - "Matplotlib: Add colorbar to non-mappable object"

Note: This example features a colorbar where the 'ticks' are placed in the middle of the color blocks. Should you be interested in something simpler, you could also use:

```
sm   = plt.cm.ScalarMappable(cmap=cmap, norm=mpl.colors.Normalize(vmin=0,vmax=2))
sm.set_array([])

cbar = fig.colorbar(sm)
cbar.set_ticks(...)
cbar.set_ticklabels(...)
```

```
import numpy             as np
import matplotlib        as mpl
import matplotlib.pyplot as plt

plt.style.use(['goose','goose-latex'])

x      = np.linspace(0, 5, 100)
N      = 21
cmap   = plt.get_cmap('jet',N)

fig,ax = plt.subplots()

for i,n in enumerate(np.linspace(0,2,N)):
  ax.plot(x, np.sin(x)*x**n, color=cmap(i))

ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')

sm   = plt.cm.ScalarMappable(cmap=cmap, norm=mpl.colors.Normalize(vmin=0,vmax=N))
sm.set_array([])

cbar = plt.colorbar(sm, ticks=np.linspace(0,N-1,(N+1)/2), boundaries=np.linspace(0,N,
↪N+1)-.5)
cbar.ax.set_yticklabels(['{0:.1f}'.format(i) for i in np.linspace(0,2,(N+1)/2)])

plt.savefig('plot-cmap.svg')
plt.show()
```

Ticks

Aligning tick labels

[source: tick-position.py]

This example is just a copy of this nice answer. See also the blog of the author.

Note: Also take note of:

```
ax.set_xticklabels(xlabels, ha='center')
ax.set_yticklabels(ylabels, va='center')
```

```
import numpy             as np
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
n       = 5
x       = np.arange(n)
y       = np.sin(np.linspace(-3,3,n))
xlabels = ['Long ticklabel %i' % i for i in range(n)]

plt.style.use(['goose', 'goose-latex'])

fig, ax = plt.subplots()

ax.plot(x,y, 'o-')

ax.set_xticks(x)

labels = ax.set_xticklabels(xlabels)

for i, label in enumerate(labels):
  label.set_y(label.get_position()[1] - (i % 2) * 0.075)

plt.savefig('tick-position.svg')
plt.show()
```

Tick formatter

[source: tick-formatter.py]

Note: References

- matplotlib.ticker
- Format Specification Mini-Language

Note: Use `matplotlib.ticker.FormatStrFormatter(...)` to use the old-style sprintf format.

```
import numpy             as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

plt.style.use(['goose', 'goose-latex'])

x = np.linspace(0,10,101)
y = x**2.

fig, ax = plt.subplots()

ax.plot(x, y)

ax.yaxis.set_major_formatter(ticker.StrMethodFormatter(r'${x:.1e}$'))
```

(continues on next page)

(continued from previous page)

```
plt.savefig('tick-formatter.svg')
plt.show()
```

Logarithmic scale

Tick rotation

[source: tick-rotation-log.py]

Specifying ticks

[source: tick-log_1.py]

Selecting ticks

[source: tick-log_2.py]

Note: References

- Selectively remove ticklabels (major and/or minor) from logarithmic axis

To have ticks at multiples of 1 and 2 of integer powers of the logarithmic base (10) use `matplotlib.ticker.LogLocator(subs=(1,2,))`. Then use `matplotlib.ticker.NullLocator()` to turn minor labels off.

1.3.2 Image

[source: image.py]

```
import numpy             as np
import matplotlib.pyplot as plt

plt.style.use(['goose','goose-latex'])

x,y  = np.meshgrid(np.linspace(0,1,100),np.linspace(0,1,100))
d    = np.sqrt(x**2+y**2)

fig,ax = plt.subplots()

cax = ax.imshow(d)
```

(continues on next page)

(continued from previous page)

```
cbar = fig.colorbar(cax,aspect=10)
cbar.set_ticks([0,np.sqrt(2.)])
cbar.set_ticklabels(['0',r'$\sqrt{2}$'])

ax.xaxis.set_ticks(range(0,101,20))
ax.yaxis.set_ticks(range(0,101,20))

ax.set_xlim([0,100])
ax.set_ylim([0,100])

ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')

plt.savefig('image.svg')
plt.show()
```

1.3.3 Colorbar

Basic

[source: image_subplots.py]

```
import numpy             as np
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

plt.style.use(['goose','goose-latex'])

# --- some data ----

a,b = np.meshgrid(np.linspace(0,1,100),np.linspace(0,1,100))
d   = np.sqrt(a**2+b**2)

# --- open figure with three axes ---

fig, axes  = plt.subplots(ncols=3, figsize=(18,6))

# --- left subplot ---

ax = axes[0]
im = ax.imshow(a,clim=(0,1))
ax.xaxis.set_ticks([0,100])
ax.yaxis.set_ticks([0,100])
ax.set_xlim([0,100])
ax.set_ylim([0,100])
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')
ax.set_title(r'$a$')
div  = make_axes_locatable(ax)
cax  = div.append_axes('right', size='5%', pad=0.1)
cbar = plt.colorbar(im,cax=cax)
```

(continues on next page)

(continued from previous page)

```
cbar.set_ticks([0,1])

# --- middle subplot ---

ax = axes[1]
im = ax.imshow(b,clim=(0,1))
ax.xaxis.set_ticks([0,100])
ax.yaxis.set_ticks([0,100])
ax.set_xlim([0,100])
ax.set_ylim([0,100])
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')
ax.set_title(r'$b$')
div  = make_axes_locatable(ax)
cax  = div.append_axes('right', size='5%', pad=0.1)
cbar = plt.colorbar(im,cax=cax)
cbar.set_ticks([0,1])

# --- right subplot ---

ax = axes[2]
im = ax.imshow(d,clim=(0,1))
ax.xaxis.set_ticks([0,100])
ax.yaxis.set_ticks([0,100])
ax.set_xlim([0,100])
ax.set_ylim([0,100])
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')
ax.set_title(r'$\sqrt{a^2 + b^2}$')
div  = make_axes_locatable(ax)
cax  = div.append_axes('right', size='5%', pad=0.1)
cbar = plt.colorbar(im,cax=cax)
cbar.set_ticks([0,1])

# --- save/show ---

plt.savefig('image_subplots.svg')
plt.show()
```

Note: References

- StackOverflow - "positioning the colorbar"

Colorbar positioning

[source: image_subplots_bottom.py]

```
import numpy             as np
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable
```

(continues on next page)

(continued from previous page)

```
plt.style.use(['goose','goose-latex'])

# --- some data ----

a,b = np.meshgrid(np.linspace(0,1,100),np.linspace(0,1,100))
d   = np.sqrt(a**2+b**2)

# --- open figure with three axes ---

fig, axes  = plt.subplots(ncols=3, figsize=(18,6))

# --- left subplot ---

ax = axes[0]
im = ax.imshow(a,clim=(0,1))
ax.xaxis.set_ticks([0,100])
ax.yaxis.set_ticks([0,100])
ax.set_xlim([0,100])
ax.set_ylim([0,100])
ax.set_title(r'$a$')
div  = make_axes_locatable(ax)
cax  = div.append_axes('bottom', size='5%', pad=0.4)
cbar = plt.colorbar(im,cax=cax, orientation='horizontal')
cbar.set_ticks([0,1])

# --- middle subplot ---

ax = axes[1]
im = ax.imshow(b,clim=(0,1))
ax.xaxis.set_ticks([0,100])
ax.yaxis.set_ticks([0,100])
ax.set_xlim([0,100])
ax.set_ylim([0,100])
ax.set_title(r'$b$')
div  = make_axes_locatable(ax)
cax  = div.append_axes('bottom', size='5%', pad=0.4)
cbar = plt.colorbar(im,cax=cax, orientation='horizontal')
cbar.set_ticks([0,1])

# --- right subplot ---

ax = axes[2]
im = ax.imshow(d,clim=(0,1))
ax.xaxis.set_ticks([0,100])
ax.yaxis.set_ticks([0,100])
ax.set_xlim([0,100])
ax.set_ylim([0,100])
ax.set_title(r'$\sqrt{a^2 + b^2}$')
div  = make_axes_locatable(ax)
cax  = div.append_axes('bottom', size='5%', pad=0.4)
cbar = plt.colorbar(im,cax=cax, orientation='horizontal')
cbar.set_ticks([0,1])

# --- save/show ---

plt.savefig('image_subplots_bottom.svg')
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

Stand-alone colorbar

[source: colorbar.py]

Note: References

- matplotlib documentation - "colorbar_only"

```
import matplotlib.pyplot as plt
import matplotlib        as mpl

plt.style.use(['goose','goose-latex'])

fig,ax = plt.subplots(figsize=(8, 2))

cbar = mpl.colorbar.ColorbarBase(ax,
  cmap        = mpl.cm.get_cmap('RdBu_r'),
  norm        = mpl.colors.Normalize(vmin=5, vmax=10),
  orientation = 'horizontal'
)

cbar.set_label('Some Units')

plt.savefig('colorbar.svg')
plt.show()
```

1.3.4 Colormap

Combined colormap

[source: colormap.py]

This example shows how to create a custom colormap. To do this one has to create an RGBA-matrix: a matrix with on each row the amount (between 0 and 1) of Red, Green, Blue, and Alpha (transparency; 0 means that the pixel does not have any coverage information and is transparent).

As an example the distance to some point is plotted in two dimensions. Then:

- For any distance higher than some critical value, the colors will be taken from a standard colormap.
- For any distance lower than some critical value, the colors will linearly go from white to the first color of the previously mentioned map.

Note: The choices depend fully on what you want to show. The colormaps and their sizes depend on your problem. For example, you can choose different types of interpolation: linear, exponential, . . . ; single- or multi-color colormaps;

etc..

Note: The choices depend fully on what you want to show. The colormaps and their sizes depend on your problem. For example you can choose a different types of interpolation: linear, exponential, ...; single- or multi-color colormaps; etc..

```
import numpy             as np
import matplotlib        as mpl
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

plt.style.use(['goose','goose-latex'])

# create colormap
# ---------------

# create a colormap that consists of
# - 1/5 : custom colormap, ranging from white to the first color of the colormap
# - 4/5 : existing colormap

# set upper part: 4 * 256/4 entries
upper = mpl.cm.jet(np.arange(256))

# set lower part: 1 * 256/4 entries
# - initialize all entries to 1 to make sure that the alpha channel (4th column) is 1
lower = np.ones((int(256/4),4))
# - modify the first three columns (RGB):
#   range linearly between white (1,1,1) and the first color of the upper colormap
for i in range(3):
  lower[:,i] = np.linspace(1, upper[0,i], lower.shape[0])

# combine parts of colormap
cmap = np.vstack(( lower, upper ))

# convert to matplotlib colormap
cmap = mpl.colors.ListedColormap(cmap, name='myColorMap', N=cmap.shape[0])

# show some example
# -----------------

# open a new figure
fig, ax = plt.subplots()

# some data to plot: distance to point at (50,50)
x,y = np.meshgrid(np.linspace(0,99,100),np.linspace(0,99,100))
z   = (x-50)**2. + (y-50)**2.

# plot data, apply colormap, set limit such that our interpretation is correct
im = ax.imshow(z, cmap=cmap, clim=(0,5000))

# add a colorbar to the bottom of the image
div  = make_axes_locatable(ax)
cax  = div.append_axes('bottom', size='5%', pad=0.4)
```

(continues on next page)

(continued from previous page)

```
cbar = plt.colorbar(im, cax=cax, orientation='horizontal')

# save/show the image
plt.savefig('colormap.svg')
plt.show()
```

Sub-colormap vs. interpolated colormap

[source: colormap-part.py]

This example contains a simple example to derive a custom colormap from an existing colormap, see this answer.

```
import numpy             as np
import matplotlib.pyplot as plt
import matplotlib        as mpl

from mpl_toolkits.axes_grid1 import make_axes_locatable

plt.style.use(['goose','goose-latex'])

np.random.seed(1)
data = np.sort(np.random.rand(8,12))

fig, axes = plt.subplots(figsize=(16,8), ncols=2)

cmap = mpl.cm.jet(np.linspace(0,1,20))
cmap = mpl.colors.ListedColormap(cmap[10:,:-1])

ax   = axes[0]
c    = ax.pcolor(data, edgecolors='k', linewidths=4, cmap=cmap, vmin=0.0, vmax=1.0)
div  = make_axes_locatable(ax)
cax  = div.append_axes('right', size='5%', pad=0.1)
cbar = plt.colorbar(c,cax=cax)

cmap = mpl.cm.jet(np.linspace(0,1,20))
cmap = mpl.colors.LinearSegmentedColormap.from_list('test', [cmap[10,:-1], cmap[-1,:-
↪1]], N=10)

ax   = axes[1]
c    = ax.pcolor(data, edgecolors='k', linewidths=4, cmap=cmap, vmin=0.0, vmax=1.0)
div  = make_axes_locatable(ax)
cax  = div.append_axes('right', size='5%', pad=0.1)
cbar = plt.colorbar(c,cax=cax)

plt.savefig('colormap-part.svg')
plt.show()
```

1.4 Examples - GooseMPL

1.4.1 Patch: plot finite element mesh

[source: patch.py]

```
import matplotlib.pyplot as plt
import GooseMPL          as gplt
import numpy             as np

from mpl_toolkits.axes_grid1 import make_axes_locatable

plt.style.use(['goose','goose-latex'])

coor = np.array([
  [ 0.0, 0.0 ],
  [ 1.0, 0.0 ],
  [ 2.0, 0.0 ],
  [ 0.0, 1.0 ],
  [ 1.0, 1.0 ],
  [ 2.0, 1.0 ],
])

conn = np.array([
  [ 0, 1, 4, 3 ],
  [ 1, 2, 5, 4 ],
])

value = np.array([
  -1,
  +1,
])

fig,ax = plt.subplots()

im = gplt.patch(coor=coor,conn=conn,cindex=value,clim=[-2,2],cmap='RdBu_r')

plt.xlim([ -0.1,  2.1 ])
plt.ylim([ -0.1,  1.1 ])

ax.set_aspect('equal')

div  = make_axes_locatable(ax)
cax  = div.append_axes("right", size="5%", pad=0.1)
cbar = plt.colorbar(im,cax=cax)

plt.savefig('patch.svg')
plt.show()
```

1.4.2 Histogram

Histogram

[source: histogram.py]

```
import matplotlib.pyplot as plt
import GooseMPL          as gplt
import numpy             as np

plt.style.use(['goose','goose-latex'])

# ----------------------------------------------------------------------------

def distribution(a=100, b=3, g=-.3, size=10000):

  r = np.random.random(size=size)

  return (a**g + (b**g - a**g)*r)**(1./g)

# ----------------------------------------------------------------------------

data = distribution()

fig, axes = plt.subplots(ncols=3, nrows=2, figsize=(3*8,2*6))

# --- histogram ---

bin_edges = gplt.histogram_bin_edges(data, bins=41)

P, x = gplt.histogram(data, bins=bin_edges, density=False)

gplt.hist(P, x, facecolor=[.2,.2,.2], axis=axes[0,0])

P, x = gplt.histogram(data, bins=bin_edges, density=True)

gplt.hist(P, x, facecolor=[.2,.2,.2], axis=axes[1,0])

# --- histogram_log ---

bin_edges = gplt.histogram_bin_edges(data, bins=41, mode='log')

P, x = gplt.histogram(data, bins=bin_edges, density=False)

gplt.hist(P, x, facecolor='b', axis=axes[0,1])

P, x = gplt.histogram(data, bins=bin_edges, density=True)

gplt.hist(P, x, facecolor='b', axis=axes[1,1])

# --- histogram_uniform ---

bin_edges = gplt.histogram_bin_edges(data, bins=41, mode='uniform')

P, x = gplt.histogram(data, bins=bin_edges, density=False)

gplt.hist(P, x, facecolor='r', axis=axes[0,2])
```

(continues on next page)

(continued from previous page)

```
P, x = gplt.histogram(data, bins=bin_edges, density=True)

gplt.hist(P, x, facecolor='r', axis=axes[1,2])

# --- axes settings ---

axes[0,0].set_title(r'histogram')
axes[0,1].set_title(r'histogram\_log')
axes[0,2].set_title(r'histogram\_uniform')

for ax in axes.ravel():

  ax.set_xlabel(r'$x$')
  ax.set_ylabel(r'$N(x)$')

plt.savefig('histogram.svg')
plt.show()
```

Probability density

[source: histogram_powerlaw.py]

```
import matplotlib.pyplot as plt
import GooseMPL          as gplt
import numpy             as np

plt.style.use(['goose','goose-latex'])

# ------------------------------------------------------------------------------

def distribution(a=100, b=3, g=-.3, size=10000):

  r = np.random.random(size=size)

  return (a**g + (b**g - a**g)*r)**(1./g)

# ------------------------------------------------------------------------------

data = distribution()

fig, axes = gplt.subplots(ncols=3)

# configure axes

for ax in axes:

  ax.set_xlabel(r'$x$')
  ax.set_ylabel(r'$\rho(x)$')

  ax.set_xscale('log')
  ax.set_yscale('log')
```

(continues on next page)

(continued from previous page)

```
  ax.set_xlim([1e+0, 1e+3])
  ax.set_ylim([1e-3, 1e+0])

# histogram

bin_edges = gplt.histogram_bin_edges(data, bins=41)

P, x = gplt.histogram(data, bins=bin_edges, density=True, return_edges=False)

axes[0].plot(x,P,marker='o', linestyle='none', markersize=5., color='k')

# histogram_log

bin_edges = gplt.histogram_bin_edges(data, bins=41, mode='log')

P, x = gplt.histogram(data, bins=bin_edges, density=True, return_edges=False)

axes[1].plot(x,P,marker='o', linestyle='none', markersize=5., color='b')

# histogram_uniform

bin_edges = gplt.histogram_bin_edges(data, bins=41, mode='uniform')

P, x = gplt.histogram(data, bins=bin_edges, density=True, return_edges=False)

axes[2].plot(x,P,marker='o', linestyle='none', markersize=5., color='r')

# add titles

axes[0].set_title(r'histogram')
axes[1].set_title(r'histogram\_log')
axes[2].set_title(r'histogram\_uniform')

# annotate powerlaw in different ways

for ax in axes:

  gplt.grid_powerlaw(exp=-1.3, axis=ax)

plt.savefig('histogram_powerlaw.svg')
plt.show()
```

1.5 Notes for developers

1.5.1 Create a new release

1. Update the version numbers by modifying *__version__* in *setup.py*.
2. Upload the changes to GitHub and create a new release there (with the correct version number).
3. Upload the package to PyPi:

```
$ python3 setup.py bdist_wheel --universal
$ twine upload dist/*
```

Note: Get `twine` by

```
python3 -m pip install --user --upgrade twine
```

Be sure to follow the possible directives about setting the user's path.

1.5.2 References

The following StackOverflow questions:

- Why using 'package_data' and not 'data_files' in 'setup.py'
- Work around to 'install' the '.mplstyle' files with the package

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index