

MPL Plotter Documentation

Antonio López Rivera

June 2022

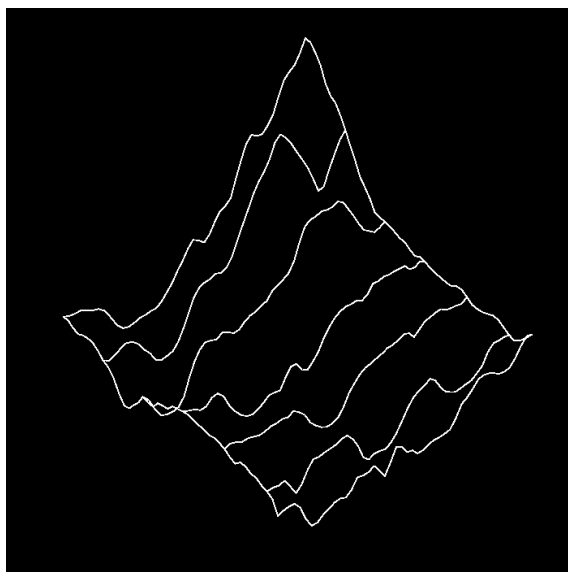




TABLE OF CONTENTS

| | | |
|----------|--|-----------|
| 1 | 2D | 2 |
| 1.1 | Composition: <code>comparison</code> | 2 |
| 1.2 | Composition: <code>panes</code> | 3 |
| 1.3 | Placeholders | 5 |
| 2 | 3D | 6 |
| 2.1 | Placeholders | 6 |
| 3 | Presets | 7 |
| 3.1 | Preset | 7 |
| 3.2 | Precision | 8 |
| 3.3 | Publication | 8 |
| 4 | Colors | 9 |
| 4.1 | Color Maps | 9 |
| 4.2 | Color Schemes | 10 |
| 4.3 | Methods | 10 |
| 5 | Internal Methods | 11 |
| 5.1 | Common | 11 |
| 5.2 | 2D Methods | 11 |
| 5.3 | 3D Methods | 12 |
| | Python Module Index | 13 |
| | Index | 14 |



Making plots for technical documents can be a time sink. MPL Plotter aims to reduce that overhead by allowing you to effortlessly and concisely

- Generate publication quality figures with a single call
- Compare data by plotting different curves in a single plot
- Visualize different kinds of data in figures with many plots

It is opinionated but built with flexibility in mind, which practically means that no default can't be changed, and any and all further customization with Matplotlib is compatible. From ticks to legends to extra axes to whatever suits your needs. There's two ways to use MPL Plotter:

- Calls to the 2D and 3D plotting functions
- Using presets, either those shipped with the library, or custom ones

It does the job for me and I expand it when it can't. Hope you find some use in it!

1.1 Composition: `comparison`

`comparison` (*x*, *y*, *f=None*, *show=False*, *autocolor=True*, *top=None*, *bottom=None*, *left=None*, *right=None*, *wspace=None*, *hspace=None*, ***kwargs*)

Data Input

The table below displays the supported numerical input combinations, where:

- `array`: List or NumPy array with numerical values
- `[...]`: List containing ...
- `result`: `<curves>`

Table 1.1: Valid input combinations.

| <i>x</i> | <i>y</i> | Result | Notes |
|-----------------------------|-----------------------------|----------|---------------------------------------|
| <code>array</code> | <code>array</code> | 1 | |
| <code>array</code> | <code>[array, array]</code> | 2 | Both <i>y</i> share a single <i>x</i> |
| <code>[array, array]</code> | <code>[array, array]</code> | 2 | Both <i>x</i> share a single <i>y</i> |
| <code>[n*[array]]</code> | <code>[n*[array]]</code> | <i>n</i> | Each <i>y</i> has an <i>x</i> |

Argument Classification

Arguments are internally classified as **figure**, **plural** and **curve** arguments, namely:

- **Figure**
Select few arguments which may be input only once in the plotting process, so as to avoid conflicts. Ieg: passing `grid=True` twice (`plt.grid(...)`) will result in no grid being drawn. These are removed from the keyword arguments and used in the last *comparison* call.
- **Plural**



Arguments passed with any of the keywords accepted by all 2D plotters -that is, any keyword which does **not** start with the name of its plotting class-, in plural tense. These must be **lists** of length equal to the **number of curves**. Each element in the list is the value of the keyword argument for each curve (eg: passing `colors=['red', 'green', 'blue']` to a 3-curve plot will set the color of the curves to 'red', 'green' and 'blue'.

- Curve

Curve-specific parameters (`color`, `line_width`, `plot_label`)

Defaults

The limits of the plot will be adjusted to the upper and lower limits of all `x`'s and `y`'s.

Parameters

- `x` (*list of list or list of np.ndarray*) – Domains.
- `y` (*list of list or list of np.ndarray*) – Values.
- `f` (*list of plot*) – Functions used to plot `y(x)`
- `autocolor` (*bool*) – Whether to automatically assign different colors to each curve
- `show` (*bool*) – `plt.show()` after plotting (thereby finishing the plot)
- `top` (*float*) – `plt.subplots_adjust` parameter
- `bottom` (*float*) – `plt.subplots_adjust` parameter
- `left` (*float*) – `plt.subplots_adjust` parameter
- `right` (*float*) – `plt.subplots_adjust` parameter
- `wspace` (*float*) – `plt.subplots_adjust` parameter
- `hspace` (*float*) – `plt.subplots_adjust` parameter
- `kwargs` – MPL Plotter plotting class keyword arguments for further customization

1.2 Composition: panes

panes (`x`, `y`, `f=None`, `fig=None`, `shape=None`, `figsize=None`, `show=False`, `rows=1`, `top=None`, `bottom=None`, `left=None`, `right=None`, `wspace=None`, `hspace=None`, `**kwargs`)



Data Input

The table below displays the supported numerical input combinations, where:

- `array`: List or NumPy array with numerical values
- `[...]`: List containing ...
- `result`: `<curves>`

Table 1.2: Valid input combinations.

| x | y | Result | Notes |
|--------------------------|--------------------------|--------|---|
| array | array | 11 | |
| array | [array, array] | 12 | Both <code>y</code> share <code>x</code> |
| [n*[array]] | [n*[array]] | 1n | Each <code>y</code> has an <code>x</code> |
| array | [array, array] | 21 | Both <code>y</code> share <code>x</code> |
| [array, array] | [array, array] | 21 | Each <code>y</code> has an <code>x</code> |
| array | [n*[array], n*[array]] | 2n | All curves in all (2) panes share a single <code>x</code> |
| [array, array] | [n*[array], n*[array]] | 2n | All curves in each pane share an <code>x</code> |
| [n*[array], n*[array]] | [n*[array], n*[array]] | 2n | All curves in all (2) panes have their own <code>x</code> |
| [n*[array], ... up to m] | [n*[array], ... up to m] | mn | All curves in all panes have their own <code>x</code> |

Argument Classification

Arguments are internally classified as **figure**, **legend**, **plural** and **curve** arguments, namely:

- Figure arguments

Arguments which may be input only once in the plotting process, so as to avoid conflicts (eg: passing `grid=True` twice (`plt.grid(...)`) will result in no grid being drawn). These are removed from the keyword arguments and applied in the last `comparison` call.

- Legend arguments

These are `plot_label/s`, which to avoid redundancy are applied in the last `comparison`. This is done only if the number of curves is the same across all panes, and equal to the number of provided `plot_labels`.

- Plural arguments

Arguments passed with any of the keywords accepted by all 2D plotters -that is, any keyword which does **not** start with the name of its plotting class-, in plural tense. These must be **lists** of length equal to the **number of panes**. Each element in the list is the value of the keyword argument for each pane (eg: `tick_labels_x=[1, 2, 3]` will set the tick labels of the x axes to 1, 2 and 3 respectively in a 3-pane plot).

- Curve arguments



Arguments passed as plurals to the comparison function. These are once more **lists** containing the value of a keyword argument, passed in plural, for each curve following the convention shown above for data input, such that passing `colors=[['red', 'blue'], ['green', 'red']]` to a plot containing 2 panes with 2 curves each will color the curves in the first pane red and blue, and those in the second green and red.

Parameters

- **x** (*list of list or list of np.ndarray or np.ndarray*) – Data
- **y** (*list of list or list of np.ndarray or np.ndarray*) – Data
- **f** (*list of function or list of plot*) – List of plotting functions to use for each curve
- **fig** (*matplotlib.figure.Figure*) – Figure object on which to plot
- **figsize** (*tuple of float*) – Figure size
- **show** (*bool*) – Whether to `plt.show()` after plotting (thereby finishing the plot)
- **rows** (*int*) – Number of rows
- **top** (*float*) – `plt.subplots_adjust` parameter
- **bottom** (*float*) – `plt.subplots_adjust` parameter
- **left** (*float*) – `plt.subplots_adjust` parameter
- **right** (*float*) – `plt.subplots_adjust` parameter
- **wspace** (*float*) – `plt.subplots_adjust` parameter
- **hspace** (*float*) – `plt.subplots_adjust` parameter
- **kwargs** – MPL Plotter plotting class keyword arguments for further customization

1.3 Placeholders

```
class MockData
```

```
    Bases: object
```

```
    filled_julia (xyz_2d=False, xyz_3d=False, df=False)
```

```
    spirograph()
```

```
    sinewave()
```

```
    waterdrop()
```

```
    boltzman (x, xmid, tau)
```

Evaluate the boltzman function with midpoint `xmid` and time constant `tau` over `x`



2.1 Placeholders

```
class MockData
    Bases: object
    waterdrop3d()
    random3d()
    hill()
```

3.1 Preset

```
class preset (plotter=None, dim=None, _dict=None)
    Bases: object
    Preset object class

    save (file)
        Save MPL Plotter preset in TOML format

    classmethod load (file)
        Load MPL Plotter preset from TOML file

class two_d (preset)
    Bases: object
    2D preset plotting methods

    class line (x=None, y=None, **kwargs)
        Bases: line

    class scatter (x=None, y=None, **kwargs)
        Bases: scatter

    class heatmap (x=None, y=None, z=None, **kwargs)
        Bases: heatmap

    class quiver (x=None, y=None, u=None, v=None, **kwargs)
        Bases: quiver

    class streamline (x=None, y=None, u=None, v=None, **kwargs)
        Bases: streamline

    class fill_area (x=None, y=None, z=None, **kwargs)
        Bases: fill_area
```



```
class three_d(preset)  
    Bases: object  
    3D preset plotting methods  
  
    class line(x=None, y=None, z=None, **kwargs)  
        Bases: line  
  
    class scatter(x=None, y=None, z=None, **kwargs)  
        Bases: scatter  
  
    class surface(x=None, y=None, z=None, **kwargs)  
        Bases: surface
```

3.2 Precision

3.3 Publication

4.1 Color Maps

custom (*red, green, blue, name='coolheat', n=1024*)

Parameters

- **red** – List of (red fraction, y0, y1) tuples
- **green** – List of (red fraction, y0, y1)
- **blue** – List of (red fraction, y0, y1)
- **name** – Colormap name
- **n** – RGB quantization levels

Returns

Colormap

mapstack (*maps, fractions=None, ranges=None*)

Create a colormap stacking an arbitrary number of conventional Matplotlib colormaps.

Parameters

- **maps** (*list of str*) – List of colormap NAMES
- **fractions** (*list of float*) – For each original colormap, the fraction it'll take of the merged colormap. [0<fr_0<1, ...]
- **ranges** (*list of tuple*) – For each original colormap, the range taken. [(0=<min<1, 0<max<=1)_0, ...]

Returns

`mpl.colors.LinearSegmentedColormap`



4.2 Color Schemes

`colorscheme_one()`

4.3 Methods

complementary (*color*, *fmt*='hex')

Return complementary of [R, G, B] or hex color.

Parameters

fmt (*string*) – Output format: 'hex' or 'rgb'.

delta (*color*, *factor*, *fmt*='hex')

Darker or lighten the input color by a percentage of <factor> ([-1, 1]) of the color spectrum (0-255).

Parameters

- **fmt** (*string*) – Output format: 'hex' or 'rgb'.
- **factor** (*float*) – [-1, 1] Measure in which the color will be modified.



INTERNAL METHODS

5.1 Common

`method_backend` (*plot*)

`method_figure` (*plot*)

`method_workspace_style` (*plot*)

`method_background_color` (*plot*)

`method_subplots_adjust` (*plot*)

`method_save` (*plot*)

`method_show` (*plot*)

5.2 2D Methods

`method_setup` (*plot*)

`method_spines` (*plot*)

`method_resize_axes` (*plot*)

`method_cb` (*plot*)

`method_grid` (*plot*)

`method_legend` (*plot*)

`method_tick_locs` (*plot*)

`method_tick_labels` (*plot*)



method_fonts (*plot*)

Reference:

- <https://matplotlib.org/2.0.2/users/customizing.html>

Pyplot method:

`plt.rcParams['<category>.<item>'] = <>`

method_title (*plot*)

method_axis_labels (*plot*)

5.3 3D Methods

method_setup (*plot*)

method_spines (*plot*)

method_pane_fill (*plot*)

method_remove_axes (*plot*)

method_scale (*plot*)

method_resize_axes (*plot*)

method_cb (*plot*)

method_grid (*plot*)

method_legend (*plot*)

method_tick_locs (*plot*)

method_tick_labels (*plot*)

method_fonts (*plot*)

Reference:

- <https://matplotlib.org/2.0.2/users/customizing.html>

Pyplot method:

`plt.rcParams['<category>.<item>'] = <>`

method_title (*plot*)

method_axis_labels (*plot*)



PYTHON MODULE INDEX

C

`mpl_plotter.color.functions`, 10
`mpl_plotter.color.maps`, 9
`mpl_plotter.color.schemes`, 9

m

`mpl_plotter.methods.common`, 11
`mpl_plotter.methods.three_d`, 12
`mpl_plotter.methods.two_d`, 11

p

`mpl_plotter.presets.precision`, 8
`mpl_plotter.presets.preset`, 7
`mpl_plotter.presets.publication`, 8

t

`mpl_plotter.three_d.mock`, 6
`mpl_plotter.two_d.comparison`, 2
`mpl_plotter.two_d.mock`, 5
`mpl_plotter.two_d.panes`, 3

B

`boltzman()` (*MockData method*), 5

C

`colorscheme_one()` (*in module `mpl_plotter.color.schemes`*), 10

`comparison()` (*in module `mpl_plotter.two_d.comparison`*), 2

`complementary()` (*in module `mpl_plotter.color.functions`*), 10

`custom()` (*in module `mpl_plotter.color.maps`*), 9

D

`delta()` (*in module `mpl_plotter.color.functions`*), 10

F

`filled_julia()` (*MockData method*), 5

H

`hill()` (*MockData method*), 6

L

`load()` (*preset class method*), 7

M

`mapstack()` (*in module `mpl_plotter.color.maps`*), 9

`method_axis_labels()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_axis_labels()` (*in module `mpl_plotter.methods.two_d`*), 12

`method_backend()` (*in module `mpl_plotter.methods.common`*), 11

`method_background_color()` (*in module `mpl_plotter.methods.common`*), 11

`method_cb()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_cb()` (*in module `mpl_plotter.methods.two_d`*), 11

`method_figure()` (*in module `mpl_plotter.methods.common`*), 11

`method_fonts()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_fonts()` (*in module `mpl_plotter.methods.two_d`*), 11

`method_grid()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_grid()` (*in module `mpl_plotter.methods.two_d`*), 11

`method_legend()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_legend()` (*in module `mpl_plotter.methods.two_d`*), 11

`method_pane_fill()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_remove_axes()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_resize_axes()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_resize_axes()` (*in module `mpl_plotter.methods.two_d`*), 11

`method_save()` (*in module `mpl_plotter.methods.common`*), 11

`method_scale()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_setup()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_setup()` (*in module `mpl_plotter.methods.two_d`*), 11

`method_show()` (*in module `mpl_plotter.methods.common`*), 11

`method_spines()` (*in module `mpl_plotter.methods.three_d`*), 12

`method_spines()` (*in module `mpl_plotter.methods.two_d`*), 11

`method_subplots_adjust()` (*in module*



`mpl_plotter.methods.common`), 11
`method_tick_labels()` (in module `mpl_plotter.methods.three_d`), 12
`method_tick_labels()` (in module `mpl_plotter.methods.two_d`), 11
`method_tick_locs()` (in module `mpl_plotter.methods.three_d`), 12
`method_tick_locs()` (in module `mpl_plotter.methods.two_d`), 11
`method_title()` (in module `mpl_plotter.methods.three_d`), 12
`method_title()` (in module `mpl_plotter.methods.two_d`), 12
`method_workspace_style()` (in module `mpl_plotter.methods.common`), 11
`MockData` (class in `mpl_plotter.three_d.mock`), 6
`MockData` (class in `mpl_plotter.two_d.mock`), 5
module
 `mpl_plotter.color`, 8
 `mpl_plotter.color.functions`, 10
 `mpl_plotter.color.maps`, 9
 `mpl_plotter.color.schemes`, 9
 `mpl_plotter.methods`, 10
 `mpl_plotter.methods.common`, 11
 `mpl_plotter.methods.three_d`, 12
 `mpl_plotter.methods.two_d`, 11
 `mpl_plotter.presets`, 6
 `mpl_plotter.presets.precision`, 8
 `mpl_plotter.presets.preset`, 7
 `mpl_plotter.presets.publication`, 8
 `mpl_plotter.three_d`, 5
 `mpl_plotter.three_d.mock`, 6
 `mpl_plotter.two_d`, 1
 `mpl_plotter.two_d.comparison`, 2
 `mpl_plotter.two_d.mock`, 5
 `mpl_plotter.two_d.panes`, 3
`mpl_plotter.color`
 module, 8
`mpl_plotter.color.functions`
 module, 10
`mpl_plotter.color.maps`
 module, 9
`mpl_plotter.color.schemes`
 module, 9
`mpl_plotter.methods`
 module, 10
`mpl_plotter.methods.common`
 module, 11
`mpl_plotter.methods.three_d`
 module, 12
`mpl_plotter.methods.two_d`
 module, 11
`mpl_plotter.presets`
 module, 6
`mpl_plotter.presets.precision`
 module, 8
`mpl_plotter.presets.preset`
 module, 7
`mpl_plotter.presets.publication`
 module, 8
`mpl_plotter.three_d`
 module, 5
`mpl_plotter.three_d.mock`
 module, 6
`mpl_plotter.two_d`
 module, 1
`mpl_plotter.two_d.comparison`
 module, 2
`mpl_plotter.two_d.mock`
 module, 5
`mpl_plotter.two_d.panes`
 module, 3

P

`panes()` (in module `mpl_plotter.two_d.panes`), 3
`preset` (class in `mpl_plotter.presets.preset`), 7

R

`random3d()` (*MockData* method), 6

S

`save()` (*preset* method), 7
`sinewave()` (*MockData* method), 5
`spirograph()` (*MockData* method), 5

T

`three_d` (class in `mpl_plotter.presets.preset`), 7
`three_d.line` (class in `mpl_plotter.presets.preset`), 8
`three_d.scatter` (class in `mpl_plotter.presets.preset`), 8
`three_d.surface` (class in `mpl_plotter.presets.preset`), 8
`two_d` (class in `mpl_plotter.presets.preset`), 7
`two_d.fill_area` (class in `mpl_plotter.presets.preset`), 7



```
two_d.heatmap (class in mpl_plotter.presets.preset),  
7  
two_d.line (class in mpl_plotter.presets.preset), 7  
two_d.quiver (class in mpl_plotter.presets.preset), 7  
two_d.scatter (class in mpl_plotter.presets.preset),  
7  
two_d.streamline (class in mpl_plotter.pre-  
sets.preset), 7
```

W

```
waterdrop () (MockData method), 5  
waterdrop3d () (MockData method), 6
```