MYPlotSpec Documentation

Release 0.1

Karl T Debiec

September 20, 2015

CHAPTER

ONE

INTRODUCTION

MYPlotSpec is a Python package used to write matplotlib-based plotting tools that support powerful configuration options using the simple text format YAML.

The goal of MYPlotSpec is to make it possible to modify plot settings such as proportions, ticks, colors, or fonts with per-figure, per-subplot, and per-dataset specificity without needing to modify Python code or implement support for individual settings. MYPlotSpec accomplishes this by parsing arguments provided in YAML format and routing them to matplotlib's existing formatting functions. MYPlotSpec should have no conflict with existing matplotlibrc settings, instead offering a level of specific control on top of them. MyPlotSpec supports a system of defaults and presets that make it easy to prepare multiple versions of plots without modifying code, such as for a lab notebook, printout, or presentation.

Sample applications of MYPlotSpec for plotting several types of data are available on GitHub:

- Dynamic Light Scattering
- Fast Protein Liquid Chromatography
- Molecular Dynamics Simulation
- Nuclear Magnetic Resonance Spectroscopy
- Ramachandran Plots

1.1 Dependencies

MYPlotSpec supports Python 2.7 and 3.4, and requires the following packages:

- matplotlib
- numpy
- six
- yaml

MYPlotSpec has been tested with Anaconda python 2.2.0 on Arch Linux, OSX Yosemite, and Windows 8.1.

1.2 Installation

Put in your `$PYTHONPATH`:

```
export PYTHONPATH=/path/to/my/python/modules:$PYTHONPATH
```

where `/path/to/my/python/modules` contains `myplotspec`.

1.3 Authorship

MYPlotSpec is developed by Karl T. Debiec, a graduate student at the University of Pittsburgh advised by Professors Lillian T. Chong and Angela M. Gronenborn.

1.4 License

CHAPTER

TWO

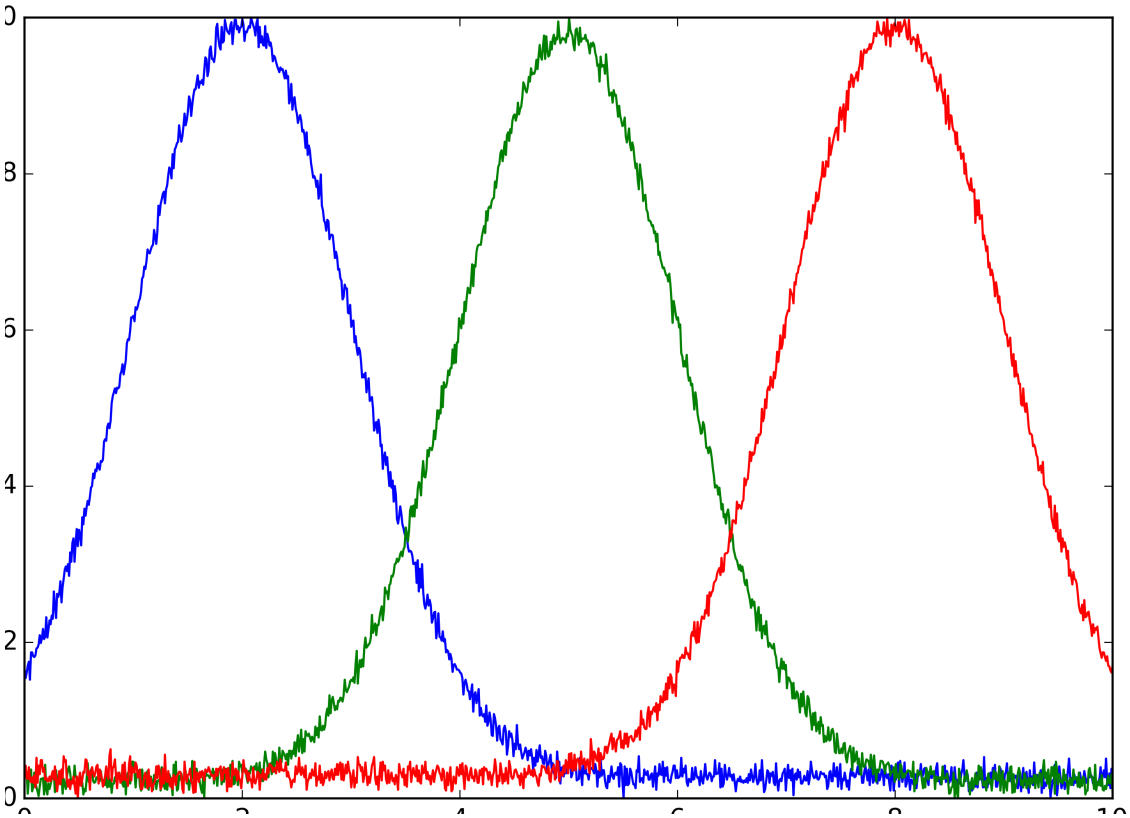
USAGE

2.1 Getting Started

2.1.1 Basic Usage

MYPlotSpec prepares matplotlib figures to specifications provided in a YAML file. At the most basic level the YAML file may specify simply an outfile and the dataset infiles to plot on it:

```
figures:
    0:
        outfile:    examples/example_1.png
        subplots:
            0:
                datasets:
                    0:
                        infile: examples/dataset_1.txt
                    1:
                        infile: examples/dataset_2.txt
                    2:
                        infile: examples/dataset_3.txt
```



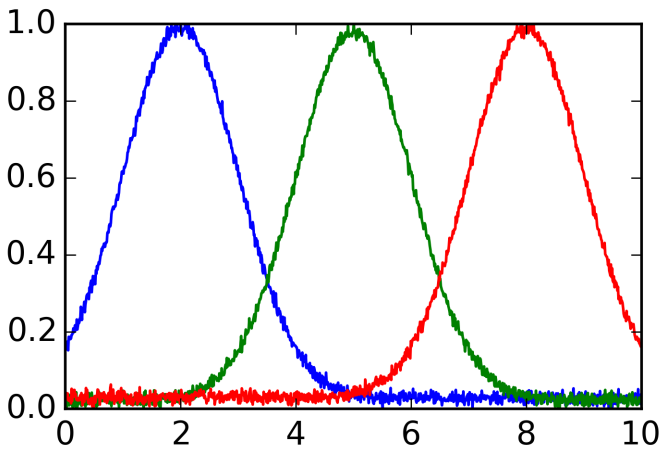
The resulting plot uses matplotlib's rcParams (i.e. defaults) for all settings, yielding a poorly-formatted figure. We could use matplotlib's functions to adjust the formatting, but we will instead specify adjustments in the YAML file, allowing MYPlotSpec to handle matplotlib for us.

2.1.2 Figure Settings

We may specify the margins and subplot dimensions of the figure:

```
figures:
    ...
    1:
        outfile:    examples/example_2.png
        left:       0.6
        sub_width:  3.0
        right:      0.2
        bottom:     0.5
        sub_height: 2.0
        top:        0.4
        subplots:
            0:
                datasets:
                    0:
                        infile: examples/dataset_1.txt
                    1:
                        infile: examples/dataset_2.txt
```

```
                    2:
                        infile: examples/dataset_3.txt
```



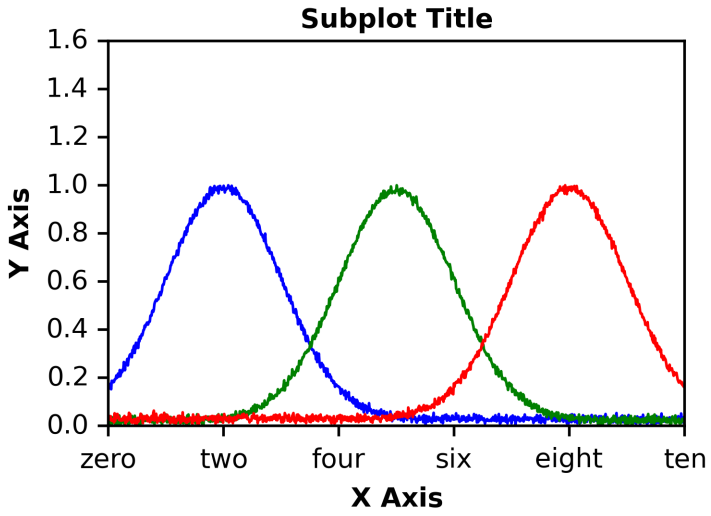
This yields a much better-proportioned figure.

2.1.3 Subplot Settings

We may further adjust the subplot formatting, adding a title and axis labels, and applying specific formatting to the ticks:

```
figures:
    ...
    2:
        outfile:    examples/example_3.png
        left:       0.6
        sub_width:  3.0
        right:      0.2
        bottom:     0.5
        sub_height: 2.0
        top:        0.4
        subplot_kw:
            autoscale_on: False
        subplots:
            0:
                title:       Subplot Title
                xlabel:      X Axis
                ylabel:      Y Axis
                xticks:      [0, 2, 4, 6, 8, 10]
                xticklabels: ["zero", "two", "four", "six", "eight", "ten"]
                yticks:      [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6]
                title_fp:    10b
                label_fp:    10b
                tick_fp:     10r
                tick_params:
                    width:     1
                    direction: out
                    top:       off
```

```
            right:     off
        datasets:
            0:
                infile: examples/dataset_1.txt
            1:
                infile: examples/dataset_2.txt
            2:
                infile: examples/dataset_3.txt
```



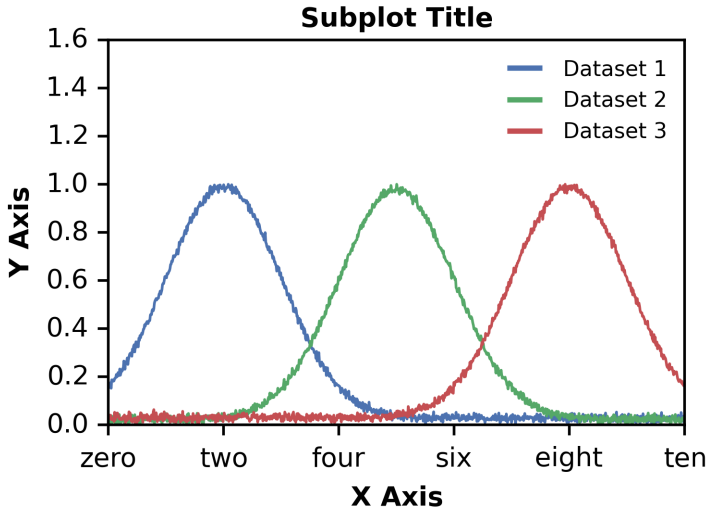
Note the setting `subplot_kw: autoscale_on: False`, this stops matplotlib from changing the bounds of the x and y axis automatically to fit the plotted data; MYPlotSpec will instead set them based on the x and y ticks. Note also how font styles may be set using a string '##L', which '##' is the font size and 'L' is 'r' for regular and 'b' for bold.

2.1.4 Dataset Settings

We may further adjust the dataset formatting, and add a legend:

```
figures:
    ...
    3:
        outfile:    examples/example_4.png
        left:       0.6
        sub_width:  3.0
        right:      0.2
        bottom:     0.5
        sub_height: 2.0
        top:        0.4
        subplot_kw:
            autoscale_on: False
        subplots:
            0:
                title:       Subplot Title
                xlabel:      X Axis
                ylabel:      Y Axis
                xticks:      [0, 2, 4, 6, 8, 10]
                xticklabels: ["zero", "two", "four", "six", "eight", "ten"]
```

```
        yticks:      [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6]
        title_fp:    10b
        label_fp:    10b
        tick_fp:     10r
        tick_params:
            width:     1
            direction: out
            top:       off
            right:     off
        legend_fp:   8r
        legend:      True
        legend_lw:   2
        legend_kw:
            frameon: False
        datasets:
            0:
                label:  Dataset 1
                infile: examples/dataset_1.txt
                color:  blue
            1:
                label:  Dataset 2
                infile: examples/dataset_2.txt
                color:  green
            2:
                label:  Dataset 3
                infile: examples/dataset_3.txt
                color:  red
```

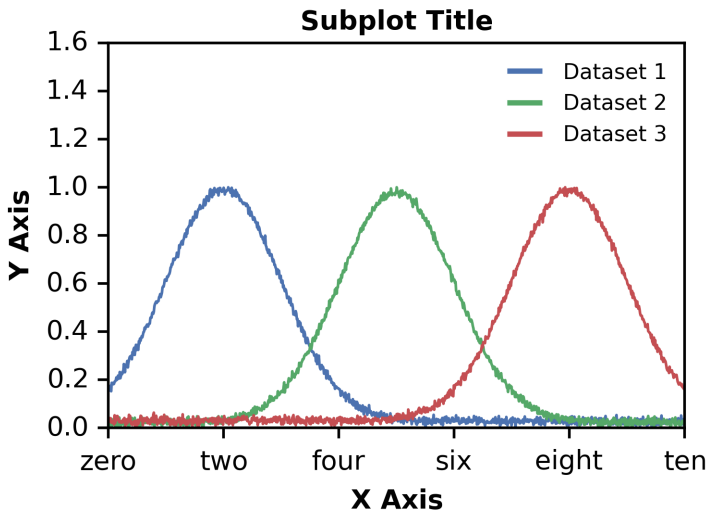


2.1.5 Presets

MYPlotSpec includes a system of presets that make it easy to switch between different plot settings without changing many settings manually. We may use the 'notebook' preset to handle to font settings for us:

```
figures:
    ...
    4:
        preset:     notebook
```

```
outfile:    examples/example_5.png
left:       0.6
sub_width:  3.0
right:      0.2
bottom:     0.5
sub_height: 2.0
top:        0.4
subplot_kw:
    autoscale_on: False
subplots:
    0:
        title:       Subplot Title
        xlabel:      X Axis
        ylabel:      Y Axis
        xticks:      [0, 2, 4, 6, 8, 10]
        xticklabels: ["zero", "two", "four", "six", "eight", "ten"]
        yticks:      [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6]
        tick_params:
            width:     1
            direction: out
            top:       off
            right:     off
        legend:      True
        legend_lw:   2
        legend_kw:
            frameon: False
        datasets:
            0:
                label:  Dataset 1
                infile: examples/dataset_1.txt
                color:  blue
            1:
                label:  Dataset 2
                infile: examples/dataset_2.txt
                color:  green
            2:
                label:  Dataset 3
                infile: examples/dataset_3.txt
                color:  red
```



In practice, most of the settings listed above would be moved into a preset, keeping the actual information needed in each YAML file small.

2.2 Subclassing FigureManager

Once the examples above are understood, the next step towards using MYPlotSpec is typically to write a subclass of *FigureManager* for the specific type of dataset in use. Generally, only the function *draw_dataset()* needs to be overridden, in addition to the attributes *defaults* and presets. Example subclasses of *FigureManager* for plotting several types of data are available on GitHub:

- Dynamic Light Scattering
- Fast Protein Liquid Chromatography
- Molecular Dynamics Simulation
- Nuclear Magnetic Resonance Spectroscopy
- Ramachandran Plots

CHAPTER

THREE

CODE

3.1 Command-Line Tools

3.1.1 FigureManager

Generates one or more figures to specifications provided in a YAML file.

class myplotspec.FigureManager.**FigureManager**(**args*, ***kwargs*)

Manages the generation of figures.

defaults

str, dict

Default arguments to *draw_report()*, *draw_figure()*, *draw_subplot()*, and *draw_dataset()* functions, in yaml format. Outer level (of indentation or keys) provides function names, and inner level provides default arguments to each function:

```
defaults = """
    method_1:
      method_1_arg_1: 1000
      method_1_arg_2: abcd
    method_2
      method_2_arg_1: 2000
      method_2_arg_2: efgh
    ...
"""
```

available_presets

str, dict

Available sets of preset arguments to *draw_report()*, *draw_figure()*, *draw_subplot()*, and *draw_dataset()* functions, in yaml format. Outer level (of indentation or keys) provides preset names, middle level provides function names, and inner level provides arguments to pass to each function when preset is active:

```
available_presets = """
  preset_1:
    method_1:
      method_1_arg_1: 1001
      method_1_arg_2: abcde
    method_2
      method_2_arg_1: 2001
      method_2_arg_2: efghi
  preset_2:
    method_1:
```

```
            method_1_arg_1: 1002
            method_1_arg_2: abcdef
          method_2
            method_2_arg_1: 2002
            method_2_arg_2: efghij
        """
```

Each preset may additionally contain the keys 'help', 'class', 'extends' and 'inherits'. 'help' may contain a short help message displayed with the help text of the script, while 'class' is the name of the category in which the preset is classified; built-in categories include 'content', for presets related to the type of data being used, 'appearance', for presets related to modifications to formatting, and 'target', for presets specifying the target destination of the figure, such as 'notebook' or 'presetnation'. 'extends' may contain the name of another preset within the class from which the preset will inherit (and optionally override) all arguments. Subclasses of this base *FigureManager* class may also include the keyword 'inherits' which may contain the name of a preset of *FigureManager* (listed below) from which it will inherit (and optionally override) all arguments.

FigureManager.**draw_report**(...)

Draws one or more figures based on provided specifications.

Figure specs are provided in a dict structured as follows:

```
figures = {
    'all': {
        'shared_xlabel': 'Time',
        'shared_ylabel': 'Measurement',
        'shared_legend': True,
        ...
    },
    '0': {
        'title': 'Trial 1',
        'subplots': {
            ...
        },
        ...
    },
    '1': {
        'title': 'Trial 2',
        'subplots': {
            ...
        },
        ...
    },
    '2': {
        'title': 'Trial 3',
        'subplots': {
            ...
        },
        ...
    },
    ...
}
```

The values stored at each (0-indexed) integer key provide the arguments to be passed to *draw_figure()* for each of a series of figures. Values stored at 'all' are passed to each figure, but overridden by values specific to that figure.

Parameters

- **figure[s]** (*dict*) – Figure specifications
- **preset[s]** (*str, list, optional*) – Selected preset(s); presets loaded from figure specification will take precedence over those passed as arguments
- **yaml_spec** (*str, dict, optional*) – Argument data structure; may be string path to yaml file, yaml-format string, or dictionary
- **verbose** (*int*) – Level of verbose output
- **debug** (*int*) – Level of debug output
- **kwargs** (*dict*) – Additional keyword arguments

Note: This function is one of two responsible for managing the output of figures to pdf files, if specified. While other output formats are single-page, pdf files may be multi-page. In order to allow multiple figures to be output to multiple pdfs, this function maintains a dict outfiles containing references to a PdfPages object for each specified pdf outfile. *draw_figure()*'s decorator *manage_output* adds new PdfPages objects as requested, or adds pages to existing ones. Once all figures have been drawn, this function closes each PdfPages.

FigureManager.**draw_figure**(...)

Draws a figure.

Figure will typically contain one or more subplots, whose specifications are provided in a dict structured as follows:

```
subplots = {
    'all': {
        'legend': True,
        ...
    },
    '0': {
        'title':    'Subplot 1',
        'datasets': {
            ...
        },
        ...
    },
    '1': {
        'title':    'Subplot 2',
        'datasets': {
            ...
        },
        ...
    },
    '2': {
        'title':    'Subplot 3',
        'datasets': {
            ...
        },
        ...
    },
    ...
}
```

The values stored at each integer key (0-indexed) provide the arguments to be passed to *draw_subplot()* for each of a series of subplots. Values stored at 'all' are passed to each subplot, but overridden by values specific to that subplot.

Figure may be annotated by drawing a title, shared x axis label, shared y axis label, or shared legend. Title and

shared axis labels are (by default) centered on all subplots present on the figure, Shared legend is drawn on an additional subplot created after those specified in subplots, using the arguments provided in 'shared_legend'.

Parameters

- **outfile** (*str*) – Output filename
- **subplot[s]** (*dict*) – Subplot specifications
- **preset[s]** (*str, list, optional*) – Selected preset(s); presets loaded from figure specification will take precedence over those passed as arguments
- **title** (*str, optional*) – Figure title
- **shared_xlabel** (*str, optional*) – X label to be shared among subplots
- **shared_ylabel** (*str, optional*) – Y label to be shared among subplots
- **shared_legend** (*bool, optional*) – Generate a legend shared between subplots
- **multiplot** (*bool, optional*) – Subplots in specification are a small multiple set; x and y labels and ticklabels are omitted for plots other than those along the left side and bottom
- **nrows** (*int, optional*) – Number of rows; only necessary with multiplot
- **ncols** (*int, optional*) – Number of columns, only necessary with multiplot
- **nsubplots** (*int, optional*) – Number of subplots; only necessary with multiplot
- **multi_xticklabels** (*list, optional*) – x tick labels to be assigned to subplots along bottom; only necessary with multiplot
- **multi_yticklabels** (*list, optional*) – y tick labels to be assigned to subplots along left side; only necessary with multiplot
- **verbose** (*int*) – Level of verbose output
- **debug** (*int*) – Level of debug output
- **kwargs** (*dict*) – Additional keyword arguments

Returns *(figure)* – Figure

FigureManager.**draw_subplot**(...)

Draws a subplot.

Subplot will typically plot one or more datasets, whose specifications are provided in a dict structured as follows:

```
    datasets = {
        'all': {
            'lw': 2,
            ...
        },
        '0': {
            'label':  'Dataset 1',
            'infile': '/path/to/dataset_1.txt',
            'color':  'red',
            ...
        },
        '1': {
            'label':  'Dataset 2',
            'infile': '/path/to/dataset_2.txt',
            'color':  'green',
            ...
        },
        '2': {
```

```
            'label':  'Dataset 3',
            'infile': '/path/to/dataset_3.txt',
            'color':  'blue',
            ...
        },
        ...
    }
```

The values stored at each integer key (0-indexed) provide the arguments to be passed to *draw_dataset()* for each of a series of datasets. Values stored at 'all' are passed to each dataset, but overridden by values specific to that dataset.

Subplot may be formatted by adjusting or labeling the x and y axes, or drawing a title or a legend.

Parameters

- **subplot** (*Axes*) – Axes on which to act
- **dataset[s]** (*dict*) – Dataset specifications
- **preset[s]** (*str, list, optional*) – Selected preset(s); presets loaded from figure specification will take precedence over those passed as arguments
- **title** (*str, optional*) – Subplot title
- **legend** (*bool, optional*) – Draw legend on subplot
- **partner_subplot** (*bool, optional*) – Add a parter subplot
- **shared_handles** (*OrderedDict, optional*) – Nascent OrderedDict of [labels]:handles shared among subplots of host figure; used to draw shared legend on figure
- **visible** (*bool, optional*) – Subplot visibility
- **verbose** (*int*) – Level of verbose output
- **debug** (*int*) – Level of debug output
- **kwargs** (*dict*) – Additional keyword arguments

FigureManager.**draw_dataset**(...)

Draws a dataset on a subplot.

Parameters

- **subplot** (*Axes*) – Axes on which to draw
- **infile** (*str*) – Path to input text file; first column is x, second is y
- **label** (*str, optional*) – Dataset label
- **color** (*str, list, ndarray, float, optional*) – Dataset color
- **plot_kw** (*dict, optional*) – Additional keyword arguments passed to subplot.plot()
- **handles** (*OrderedDict, optional*) – Nascent OrderedDict of [labels]: handles on subplot
- **verbose** (*int*) – Level of verbose output
- **debug** (*int*) – Level of debug output
- **kwargs** (*dict*) – Additional keyword arguments

FigureManager.**load_dataset**(...)

Loads a dataset, or reloads a previously-loaded dataset.

Datasets are stored in *FigureManager*'s dataset_cache instance variable, a dictionary containing copies of previously loaded datasets keyed by tuples containing the class and arguments used to instantiate the dataset.

In order to support caching, a class must implement the static method 'get_cache_key', which generates the tuple key. Only arguments that affect the resulting dataset should be included in the key (e.g. 'infile' should be included, but 'verbose' and 'debug' should not).

Cachable dataset classes may also implement the method 'get_cache_message' which returns a message to display when the dataset is loaded from the cache.

Parameters

- **cls** (*class*) – Dataset class
- **args** (*tuple*) – Positional arguments passed to cls.get_cache_key() and cls()
- **kwargs** (*dict*) – Keyword arguments passed to cls.get_cache_key() and cls()

Returns **dataset** (*cls*) – Dataset, either initialized new or copied from cache

FigureManager.**initialize_presets**(...)
Initializes presets.

Parameters

- **available_presets** (*string, dict*) – Available presets; may be a yaml string, path to a yaml file, or a dictionary; if not provided pulled from *available_presets*
- **args** (*tuple*) – Additional positional arguments
- **kwargs** (*dict*) – Additional keyword arguments

FigureManager.**main**(...)
Provides command-line functionality.

Parameters **parser** (*ArgumentParser*) – Argparse argument parser; enables sublass to instantiate parser and add arguments (optional)

3.2 Decorators

Note: MyPlotSpec's decorator classes all support arguments provided at decoration (e.g. `@decorator(foo = bar)`). These use a different syntax from decorator classes without arguments. When the wrapped function is declared, `__init__` and `__call__` from the decorator are called sequentially. `__init__` receives the arguments, while `__call__` receives the function. `__init__` should store the values of the arguments, while `__call__` should prepare and return a wrapped function using their values. Subsequent calls will go to the wrapped function. For decorator classes without arguments, `__init__` is called when the function is declared, and should store the reference to the function; `__call__` is called when the function is called, and should carry out the pre-function decorator logic, run the function, and carry out the post-function decorator logic.

Note: MyPlotSpec's decorator classes *manage_kwargs*, *manage_output*, and *debug_arguments* may be used to wrap either functions or methods. This is enabled by restricting the arguments of their `wrapped_function` to `*args` and `**kwargs`, and accessing any arguments needed by the decorator using `kwargs.pop()` or `kwargs.get()`. If a method is wrapped, the first argument is the host object of the method (`self`), shifting the positions of other named arguments.

3.2.1 manage_defaults_presets

Decorator to manage the passage of defaults and available presets to a method.

class myplotspec.manage_defaults_presets.**manage_defaults_presets**(*verbose=1*, *debug=0*)

Decorator to manage the passage of defaults and available presets to a method.

This decorator is a partner to *manage_kwargs*, desiged to allows its use for methods of objects containg central `defaults` and `available_presets` attributes. It obtains available defaults and presets for the wrapped method from their central location in the host object, and passes on those applicable to the wrapped method. *manage_kwargs* then selects arguments to pass from among the provided defaults, available and selected presets, YAML file, and arguments provided at call time.

Defaults are accessed from the host object's instance (or class) variable `self.defaults`, and may be a dict, a path to a YAML file, or a YAML string. Outer level (of indentation or keys) provides function names, and inner level provides default arguments to each function:

```
defaults = """
    method_1:
        method_1_arg_1: 1000
        method_1_arg_2: abcd
    method_2
        method_2_arg_1: 2000
        method_2_arg_2: efgh
    ...
"""
```

Presets are accessed from the host objects's instance (or class) variable `self.available_presets`, in the same formats as `self.defaults`. Presets contain an outer level of keys providing the names of available presets:

```
available_presets = """
    preset_1:
        method_1:
            method_1_arg_1: 1001
            method_1_arg_2: abcde
        method_2
            method_2_arg_1: 2001
            method_2_arg_2: efghi
    preset_2:
        method_1:
            method_1_arg_1: 1002
            method_1_arg_2: abcdef
        method_2
            method_2_arg_1: 2002
        method_2_arg_2: efghij
"""
```

When this decorator is used to wrap a method of a class, it adds to the arguments being passed `defaults`, containing the defaults specified for the method, and `available_presets`, containing only the presets applicable to the method:

```
@manage_defaults_presets()
def method_1(*args, **kwargs):
    print(kwargs)
    ...

{
    'defaults': {
```

```
            'method_1_argument_1': 1000,
            'method_1_argument_2': 'asdf'
        },
        'presets': {
            'preset_1': {
                'method_1_argument_1': 1001,
                'method_1_argument_2': 'asde'
            },
            'preset_1': {
                'method_1_argument_1': 1002,
                'method_1_argument_2': 'asdef'
            }
        },
        ...
    }
```

verbose
int

Level of verbose output

debug
int

Level of debug output

3.2.2 manage_kwargs

Decorator to manage the passage of keyword arguments to a function or method.

class myplotspec.manage_kwargs.**manage_kwargs**(*verbose=0*, *debug=0*)
Decorator to manage the passage of keyword arguments to a function or method.

Accumulates keyword arguments from several sources, in order of increasing priority:

1. Defaults

   Obtained from the argument `defaults`, which may be a dict, a path to a YAML file, or a YAML string:

```
    my_function(
        defaults = {
            'fig_width':  5.0
            'fig_height': 5.0
        },
        ...
    )
```

2. Presets

   Available presets are obtained from the argument `available_presets`, which may be a dict, a path to a YAML file, or a YAML string. Selected presets are obtained from the argument `presets`, which may be a string or list of strings in order of increasing priority:

```
    my_function(
        presets = 'letter',
        available_presets = {
            'letter': {
```

```
                    'fig_width':   8.5
                    'fig_height': 11.0
                },
                'legal': {
                    'fig_width':   8.5
                    'fig_height': 14.0
                }
            },
            ...
        )
```

3.YAML file

YAML file is obtained from the keyword argument `yaml_spec`, which may be a dict, a path to a YAML file, or a YAML string. Selected keys within the YAML file from which to load arguments are obtained from the argument `yaml_keys`, which is a list of lists in order of increasing priority:

```
        my_function(
            yaml_spec = {
                'figures': {
                    'all': {
                        'fig_width':  11.0,
                        'fig_height': 17.0,
                        'outfile':    'plot.pdf'
                    },
                    '0': {
                        'fig_width':  12.0
                    }
                }
            },
            yaml_keys = [['figures', 'all'], ['figures', '0']],
            ...
        )
```

If `yaml_keys` is omitted, the complete yaml file will be used.

4.Function call

Arguments provided at function call:

```
        my_function(
            fig_width  = 6.0,
            fig_height = 6.0,
            ...
        )
```

All of the above will override defaults provided in the function declaration itself.

verbose
int

Level of verbose output

debug
int

Level of debug output

3.2.3 manage_output

Decorator to manage the output of matplotlib figures.

class myplotspec.manage_output.**manage_output**(*verbose=False*, *debug=False*)
Decorator to manage the output of matplotlib figures.

Saves figure returned by wrapped function to a file named `outfile`; passing additional keyword arguments `savefig_kw` to `Figure.savefig()`. For pdf output, additional argument `outfiles` may be provided; containing a dictionary whose keys are the paths to output pdf files, and whose values are open PdfPages objects representing those files. The purpose of this is to allow figures output from multiple calls to the wrapped function to be output to sequential pages of the same pdf file. Typically `outfiles` will be initialized before calling this wrapped function; and once calls to the function is complete the `PdfPages.close()` method of each outfile in `outfiles` is called.

3.3 Functions

3.3.1 General

General functions.

myplotspec.**get_yaml**(*input*)
Generates a data structure from yaml input.

Parameters **input** (*str, dict*) – yaml input; if str, tests whether or not it is a path to a yaml file. If it is, the file is loaded using yaml; if it is not a file, the string itself is loaded using yaml. If dict, returned without modification

Returns *output* – Data structure specified by input

Warns **UserWarning** – Loaded data structure is a string; occurs if input file is a path to a yaml file that is not found

Raises `TypeError` – Input is not str or dict

myplotspec.**merge_dicts**(*dict_1*, *dict_2*)
Recursively merges two dictionaries.

Parameters

- **dict_1** (*dict*) – First dictionary
- **dict_2** (*dict*) – Second dictionary; values for keys shared by both dictionaries are drawn from dict_2

Returns *(dict)* – Merged dictionary

myplotspec.**multi_get**(**args*, ***kwargs*)
Scans dict for keys; returns first value.

Parameters

- **keys** (*str, list*) – Acceptable key(s) in order of decreasing priority
- **dictionary** (*dict*) – dict to be tested
- **value** – Default to be returned if not found

Returns *value* – Value from first matching key; or None if not found

myplotspec.**multi_get_copy**(**args*, ***kwargs*)
Scans dict for keys; returns copy of first value.

Parameters

- **keys** (*str, list*) – Acceptable key(s) in order of decreasing priority
- **dictionary** (*dict*) – dict to be tested
- **value** – Default to be returned if not found

Returns *value* – Value from first matching key; or None if not found

myplotspec.**multi_pop**(**args*, ***kwargs*)
Scans dict for keys; returns first value and deletes it and others.

Parameters

- **keys** (*str, list*) – Acceptable key(s) in order of decreasing priority
- **dictionary** (*dict*) – dict to be tested
- **value** – Default to be returned if not found

Returns *value* – Value from first matching key; or None if not found

myplotspec.**pad_zero**(*ticks*, *digits=None*, ***kwargs*)
Prepares list of tick labels, each with the same precision.

Parameters

- **ticks** (*list, ndarray*) – ticks
- **digits** (*int, optional*) – Precision; by default uses largest provided

Returns **tick_labels** (*list*) – Tick labels, each with the same number of digits after the decimal point

3.3.2 matplotlib

General

myplotspec.**get_color**(*color*)
Generates a color.

If color is a str, may be of form 'pastel.red', 'dark.blue', etc. corresponding to a color set and color; if no set is specified the 'default' set is used. If list or ndarray, should contain three floating point numbers corresponding to red, green, and blue values. If these numbers are greater than 1, they will be divided by 255. If float, should correspond to a grayscale shade, if greater than 1 will be divided by 255.

Parameters **color** (*str, list, ndarray, float*) – color

Returns *(list)* – [red, green, blue] on interval 0.0-1.0

myplotspec.**get_edges**(*figure_or_subplots*, ***kwargs*)
Finds the outermost edges of a set of subplots on a figure.

Parameters **figure_or_subplots** (*Figure, list, dict*) – Axes whose edges to get; if Figure, use all Axes

Returns *(dict)* – Edges; keys are 'left', 'right', 'top', and 'bottom'

myplotspec.**get_figure_subplots**(*figure=None*, *subplots=None*, *nrows=None*, *ncols=None*, *nsubplots=None*, *left=None*, *sub_width=None*, *wspace=None*, *right=None*, *bottom=None*, *sub_height=None*, *hspace=None*, *top=None*, *fig_width=None*, *fig_height=None*, *figsize=None*, *verbose=1*, *debug=0*, ***kwargs*)

Generates a figure and subplots to provided specifications.

Differs from matplotlib's built-in functions in that it:

- Accepts subplot dimensions is inches rather than proportional figure coordinates
- Optionally calculates figure dimensions from provided subplot dimensions, rather than the reverse
- Returns subplots in an OrderedDict
- Smoothly adds additional subplots to a previously-generated figure (i.e. can be called multiple times)

Parameters

- **figure** (*Figure, optional*) – Figure, if adding subplots to a preexisting Figure
- **subplots** (*OrderedDict, optional*) – Subplots, if adding subplots to a prevexisting Figure
- **nrows** (*int*) – Number of rows of subplots to add
- **ncols** (*int*) – Number of columns of subplots to add
- **nsubplots** (*int, optional*) – Number of subplots to add; if less than nrows*ncols (e.g. 2 cols and 2 rows but only three subplots)
- **sub_width** (*float*) – Width of subplot(s)
- **sub_height** (*float*) – Height of subplot(s)
- **left** (*float*) – Margin between left side of figure and leftmost subplot
- **right** (*float*) – Margin between right side of figure and rightmost subplot
- **top** (*float*) – Margin between top of figure and topmost subplot
- **bottom** (*float*) – Margin between bottom of figure and bottommost subplot
- **wspace** (*float*) – Horizontal margin between adjacent subplots
- **hspace** (*float*) – Vertical margin between adjacent subplots
- **fig_width** (*float*) – Width of figure; by default calculated from left, sub_width, wspace, right, and ncols
- **fig_height** (*float*) – Height of figure, by default calculated from bottom, sub_height, hspace, top, and nrows
- **figsize** (*list*) – Equivalent to [`fig_width`, `fig_height`]
- **figure_kw** (*dict*) – Additional keyword arguments passed to figure()
- **subplot_kw** (*dict*) – Additional keyword arguments passed to Axes()
- **axes_kw** (*dict*) – Alias to `subplot_kw`
- **verbose** (*int*) – Level of verbose output
- **debug** (*int*) – Level of debug output
- **kwargs** (*dict*) – Additional keyword arguments

Returns *(*Figure, OrderedDict)** – Figure and subplots

myplotspec.**get_font**(*fp=None*, ***kwargs*)
Generates font based on provided specification.

fp may be a string of form '##L' in which '##' is the font size and L is 'r' for regular or 'b' for bold. fp may also be a dict of keyword arguments to pass to FontProperties. *fp* may also be a FontProperties, in which case it is returned without modification.

Parameters **fp** (*str, dict, FontProperties*) – Font specifications

Returns *(FontProperties)* – Font with given specifications

Axes

Functions for formatting axes.

myplotspec.axes.**set_xaxis**(*subplot*, *xticks=None*, *xticklabels=None*, *xtick_fp=None*, *tick_fp=None*, *xticklabel_fp=None*, *ticklabel_fp=None*, *xlabel=None*, *xlabel_fp=None*, *label_fp=None*, *xtick_params=None*, *tick_params=None*, *xlw=None*, *lw=None*, ***kwargs*)
Formats the x axis of a subplot using provided keyword arguments.

Parameters

- **subplot** (*Axes*) – Axes to format
- **xticks** (*list or ndarray*) – Ticks; first and last are used as upper and lower boundaries
- **xtick_kw** (*dict*) – Keyword arguments passed to subplot.set_xticks()
- **xticklabels** (*list*) – Tick label text
- **[x]tick[label]_fp** (*str, dict, FontProperties*) – Tick label font
- **xticklabel_kw** (*dict*) – Keyword arguments passed to subplot.set_xticklabels()
- **xlabel** (*str*) – Label text
- **[x]label_fp** (*str, dict, FontProperties*) – Label font
- **xlabel_kw** (*dict*) – Keyword arguments passed to subplot.set_xlabel()
- **[x]tick_params** (*dict*) – Keyword arguments passed to subplot.tick_params(); only affect x axis
- **[x]lw** (*float*) – Subplot top and bottom line width
- **kwargs** (*dict*) – Additional keyword arguments

myplotspec.axes.**set_yaxis**(*subplot*, *subplot_y2=None*, *yticks=None*, *y2ticks=None*, *yticklabels=None*, *y2ticklabels=None*, *ytick_fp=None*, *y2tick_fp=None*, *tick_fp=None*, *yticklabel_fp=None*, *y2ticklabel_fp=None*, *ticklabel_fp=None*, *ylabel=None*, *y2label=None*, *ylabel_fp=None*, *y2label_fp=None*, *label_fp=None*, *ytick_params=None*, *y2tick_params=None*, *tick_params=None*, *ylw=None*, *lw=None*, ***kwargs*)
Formats the y axis of a subplot using provided keyword arguments.

Parameters

- **subplot** (*Axes*) – Axes to format
- **subplot_y2** (*Axes, optional*) – Second y axes to format; if this is omitted, but y2ticks is included, the second y axis will be generated
- **yticks** (*list or ndarray*) – Ticks; first and last are used as upper and lower boundaries

- **ytick_kw** (*dict*) – Keyword arguments passed to subplot.set_yticks()
- **yticklabels** (*list*) – Tick label text
- **[y]tick[label]_fp** (*str, dict, FontProperties*) – Tick label font
- **yticklabel_kw** (*dict*) – Keyword arguments passed to subplot.set_yticklabels()
- **ylabel** (*str*) – Label text
- **[y]label_fp** (*str, dict, FontProperties*) – Label font
- **ylabel_kw** (*dict*) – Keyword arguments passed to subplot.set_ylabel()
- **[y]tick_params** (*dict*) – Keyword arguments passed to subplot.tick_params(); only affect y axis
- **[y]lw** (*float*) – Subplot top and bottom line width
- **y2ticks** (*list or ndarray*) – Ticks for second y axis; first and last are used as upper and lower boundaries; if this argument is provided, a y2 axis is generated using subplot.twiny()
- **y2tick_kw** (*dict*) – Keyword arguments passed to subplot.set_yticks() for second y axis
- **y2ticklabels** (*list*) – Tick label text for second y axis
- **[y2]tick[label]_fp** (*str, dict, FontProperties*) – Tick label font for second y axis
- **y2ticklabel_kw** (*dict*) – Keyword arguments passed to subplot.set_yticklabels() for second y axis
- **y2label** (*str*) – Label text for second y axis
- **[y2]label_fp** (*str, dict, FontProperties*) – Label font for second y axis
- **y2label_kw** (*dict*) – Keyword arguments passed to subplot.set_ylabel() for second y axis
- **kwargs** (*dict*) – Additional keyword arguments

Text

Functions for formatting text.

myplotspec.text.**set_title**(*figure_or_subplot*, *title=None*, *title_fp=None*, **args*, ***kwargs*)

Draws a title on a Figure or subplot.

Parameters

- **figure_or_subplot** (*Figure, Axes*) – Object on which to draw title
- **title** (*str*) – Title text
- **title_fp** (*str, dict, FontProperties*) – Title font
- **top** (*float*) – Distance between top of figure and title (inches); Figure title only
- **title_kw** (*dict*) – Keyword arguments passed to `Figure.suptitle()` or `Axes.set_title()`

Returns *(*Text)** – Title

myplotspec.text.**set_shared_xlabel**(*figure_or_subplots*, *xlabel=None*, *xlabel_fp=None*, *label_fp=None*, **args*, ***kwargs*)

Draws an x axis label shared by multiple subplots.

The horizontal position of the shared x label is (by default) the center of the selected subplots, and the vertical position is a specified distance from the bottom of the figure.

Parameters

- **figure_or_subplots** (*Figure, OrderedDict*) – Subplots to use to calculate label horizontal position; if Figure, all subplots present on figure are used
- **xlabel** (*str*) – Label text
- **[x]label_fp** (*str, dict, FontProperties*) – Label font
- **xlabel_kw** (*dict*) – Keyword arguments passed to *set_text()*
- **bottom** (*float*) – Distance between bottom of figure and label (inches); if negative, distance between bottommost plot and label
- **top** (*float*) – Distance between top of figure and label; if negative, distance between topmost subplot and label; overrides `bottom`
- **x** (*float*) – X position within figure (proportion 0.0-1.0); default = center of selected subplots
- **y** (*float*) – Y position within figure (proportion 0.0-1.0); overrides `bottom` and `top`

Returns *(*Text)** – X axis label

`myplotspec.text.`**set_shared_ylabel**(*figure_or_subplots*, *ylabel=None*, *ylabel_fp=None*, *label_fp=None*, **args*, ***kwargs*)

Draws a y-axis label shared by multiple subplots.

The vertical position of the shared y label is (by default) the center of the selected subplots, and the horizontal position is a specified distance from the left edge of the figure.

Parameters

- **figure_or_subplots** (*Figure, OrderedDict*) – Subplots to use to calculate label vertical position; if Figure, all subplots present on figure are used
- **ylabel** (*str*) – Label text
- **[y]label_fp** (*str, dict, FontProperties*) – Label font
- **ylabel_kw** (*dict*) – Keyword arguments passed to *set_text()*
- **left** (*float*) – Distance between left side of figure and label; if negative, distance between leftmost plot and label
- **right** (*float*) – Distance between right side of figure and label; if negative, distance between rightmost plot and label; overrides `left`
- **x** (*float*) – X position within figure (proportion 0.0-1.0); overrides `left` and `right`
- **y** (*float*) – Y position within figure (proportion 0.0-1.0); default = center of selected subplots

Returns *(*Text)** – Y axis label

`myplotspec.text.`**set_inset**(*subplot*, *inset=None*, *inset_fp=None*, **args*, ***kwargs*)

Draws a text inset on a subplot.

Parameters

- **subplot** (*Axes*) – Subplot on which to draw inset
- **inset** (*str*) – Inset text
- **inset_fp** (*str, dict, FontProperties*) – Inset font
- **inset_kw** (*dict*) – Keyword arguments passed to *set_text()*
- **xpro** (*float*) – X position within subplot (proportion 0.0-1.0)
- **ypro** (*float*) – Y position within subplot (proportion 0.0-1.0)

- **x** (*float*) – X position within subplot (subplot coordinate); overrides `xpro`
- **y** (*float*) – Y position within subplot (subplot coordinate), overrides `ypro`

Returns *(*Text)** – Inset text

`myplotspec.text.`**`set_text`**(*figure_or_subplot*, *s=None*, *x=None*, *y=None*, *text=None*, *fp=None*, *border_lw=None*, **args*, ***kwargs*)

Prints text on a figure or subplot.

Parameters

- **figure_or_subplot** (*Figure, Axes*) – Object on which to draw
- **text** (*str*) – Text
- **fp** (*str, dict, FontProperties*) – Text font
- **text_kw** (*dict*) – Keyword arguments passed to `text()`

Returns *(*Text)** – Text

Legend

Functions for formatting legends.

Note: Acceptable values of `loc` and their meanings, for reference:

```
0 = Best
+---------+
|2   9   1|
|6  10   7|
|3   8   4|
+---------+
```

`myplotspec.legend.`**`set_legend`**(*subplot*, *handles=None*, *legend_lw=None*, *legend_fp=None*, ***kwargs*)

Draws and formats a legend on a subplot.

Parameters

- **subplot** (*Axes*) – Subplot to which to add legend
- **handles** (*OrderedDict*) – Collection of [labels]: handles for datasets to be plotted on legend; by default all available datasets are included
- **legend_lw** (*float*) – Legend handle linewidth
- **legend_fp** (*str, dict, FontProperties*) – Legend font
- **legend_kw** (*dict*) – Keyword arguments passed to subplot.legend()

Returns *(*Legend)** – Legend

`myplotspec.legend.`**`set_shared_legend`**(*figure*, *subplots*, ***kwargs*)

Draws a legend on a figure, shared by multiple subplots.

Useful when several plots on the same figure share the same description and plot style.

Parameters

- **figure** (*Figure*) – Figure to which to add shared legend
- **subplots** (*OrderedDict*) – Collection of subplots to which to append new subplot for shared legend

Returns *(*Legend)** – Legend

3.4 Debug

Classes and functions for debugging.

3.4.1 Decorators

class myplotspec.debug.**debug_arguments**(*debug=0*)
Decorator to debug argument passage to a function or method.

Provides more verbose output if a TypeError is encountered at function call

debug
int

Level of debug output

3.4.2 Output functions

myplotspec.debug.**db_s**(*string*, *indent=0*)
Prints debug output.

Parameters

- **string** (*str*) – Content of debug output
- **indent** (*int, optional*) – Indentation level; multiplied by 4

myplotspec.debug.**db_kv**(*key*, *value*, *indent=0*, *flag=u' '*)
Prints debug output for a key:value pair, truncated to 80 columns.

Parameters

- **key** (*str*) – key
- **value** (*str*) – value
- **indent** (*int, optional*) – Indentation level; multiplied by 4
- **flag** (*str, optional*) – Single-character flag describing pair

3.4.3 Formatting Functions

myplotspec.debug.**identify**(*subplots*, ***kwargs*)
Identifies key of each subplot with inset text.

Parameters **subplots** (*OrderedDict*) – subplots

V