mATLASplotlib Documentation

James Robinson

Jul 02, 2018

Contents

This package provides wrappers around matplotlib functionality produce plots compatible with the style guidelines for the ATLAS experiment at the LHC. It is particularly aimed at users who are not familiar with matplotlib syntax. Basic usage involves creating a canvas, plotting a dataset and saving the output to a file. For example, something like

```
import mATLASplotlib
with mATLASplotlib.canvases.Simple(shape="landscape") as canvas:
    x, y = [0, 1, 2, 3], [0, 1, 4, 9]
    canvas.plot_dataset(x, y, style="scatter", label="Example points", colour="black")
    canvas.save("simple_example")
```

will produce a minimal scatter plot with automatically determined axis limits and save this to a PDF (if not otherwise specified).

CHAPTER 1

Introduction

This package is meant to be a simple wrapper around `matplotlib` for users who want to produce plots conforming to the style guide of the ATLAS experiment at the LHC. Rather than interacting with `matplotlib` classes directly, these are managed through a *`canvas`* class which has methods related to the plotting and styling of datasets and then writing the resulting output to graphical formats on disk.

CHAPTER 2

Installation

2.1 1. Requirements

- `Python` 2.7 (not yet tested with Python 3 but it should work)
- `ROOT` with `PyROOT` enabled
- `matplotlib`
- `numpy`
- `scipy`

mATLASplotlib is developed and tested on Linux and Mac.

2.2 2. Automatic installation with pip

To install a released version of mATLASplotlib use `pip`.

To install in your home directory:

```
pip install --user mATLASplotlib
```

To install system-wide (requires root privileges):

```
sudo pip install mATLASplotlib
```

2.3 3. Upgrading with pip

To upgrade with `pip` simply do

```
pip install -U mATLASplotlib
```

running with sudo as before if this is a system-wide installation

2.4 4. Installing from source

Clone the repository with `git`:

```
git clone https://github.com/jemrobinson/mATLASplotlib.git
```

After doing this, you should use the `setup.py` script to install.

To install in your home directory:

```
python setup.py install --user
```

To install system-wide (requires root privileges):

```
sudo python setup.py install
```

CHAPTER 3

Font setup

To allow `matplotlib` to use Helvetica (required for ATLAS style) follow the guidelines from here https://olgabotvinnik.com/blog/how-to-set-helvetica-as-the-default-sans-serif-font-in/ (reproduced below).

3.1 1. Linux users

For Linux users, adding the `.ttf` fonts to `~/.fonts` and removing `~/.matplotlib/fontList.cache` `~/.matplotlib/fontManager.cache` `~/.matplotlib/ttffont.cache` should be enough.

3.2 2. OSX users

- Download and install `fondu` to convert Mac-Helvetica to ttf-Helvetica

```
brew install fondu
```

- Find Helvetica on your system

Probably somewhere like: `/System/Library/Fonts/Helvetica.dfont`

- Find where `matplotlib` stores its data
- Start a `python` prompt and run:

```
import matplotlib; matplotlib.matplotlib_fname()
```

and get output like:

```
u'/usr/local/lib/python2.7/site-packages/matplotlib/mpl-data/matplotlibrc'
```

we need to put the `.ttf` from this path in `fonts/ttf`

- Create the `.ttf`

```
sudo fondu -show /System/Library/Fonts/Helvetica.dfont
```

- Edit your `.matplotlibrc` file

Edit `~/.matplotlib/matplotlibrc`, find the line `font.sans-serif : Bitstream Vera Sans, ...` and put `Helvetica` at the beginning of this list

- Force matplotlib to re-scan the font lists

Now we need to force matplotlib to re-create the font lists by removing the files.

```
rm ~/.matplotlib/fontList.cache ~/.matplotlib/fontManager.cache ~/.matplotlib/ttffont.
↪cache
```

CHAPTER 4

Getting Started

4.1 1. Validate installation

After following the installation instructions, `mATLASplotlib` should be available at the `python` prompt

```
>>> import mATLASplotlib
```

The user interface of `mATLASplotlib` is centered on a *`canvas`* on which datasets can be plotted.

4.2 2. Constructing some data

To demonstrate how plotting works, we need some data: let's construct some using `ROOT` and `numpy`

```
import numpy as np
import ROOT
hist = ROOT.TH1F("Generated data", "This is some autogenerated data", 40, -4, 4)
for x in np.random.normal(size=10000):
    hist.Fill(x)
```

this should have drawn 10000 samples from a normal distribution and added them to a ROOT histogram.

4.3 3. Setting up a canvas

We use a context manager to open the canvas, which ensures that necessary cleanup is done when the canvas is no longer needed. Currently the supported canvases are the *`Simple`* canvas which contains one set of `matplotlib` axes, the *`Ratio`* canvas, which contains a main plot and a ratio plot underneath, and the the *`Panelled`* canvas which contains a top panel and an arbitrary number of lower panels beneath it.

```
import mATLASplotlib
with mATLASplotlib.canvases.Simple(shape="square") as canvas:
    canvas.plot_dataset(hist, style="scatter", label="Generated data", colour="black")
```

The three shapes preferred by the ATLAS style guide are "square" (600 x 600 pixels), "landscape" (600 x 800 pixels) and "portrait" (800 x 600 pixels). Here we have chosen to use "square".

After setting up the canvas, we can plot the dataset we constructed earlier using the *plot_dataset* method.

4.4 4. Plotting options

The different `style` options specify how the data should be displayed. Options are

- *bar* (a histogram or bar chart)
- *binned_band* (a band with a fill colour in between the maximum and minimum values in each bin)
- *coloured_2D* (a 2D histogram with a colour-scale to indicate the 'z' value in each bin)
- *line* (a single line, either smooth or consisting of straight line segments)
- *scatter* (a scatter plot - often used for data points)
- *stack* (one of a series of histograms that should be summed up when drawn)

Other options like `linestyle` and `colour` can be used to distinguish different datasets.

4.5 5. Saving the canvas to a file

Saving the output to a file is very simple.

```
canvas.save("simple_example")
```

This function takes an optional `extension` argument which sets the file extension of the output file. Running this code will produce a minimal scatter plot with automatically determined axis limits and save this to a PDF (if not otherwise specified). The output should be similar to that shown in the image below.

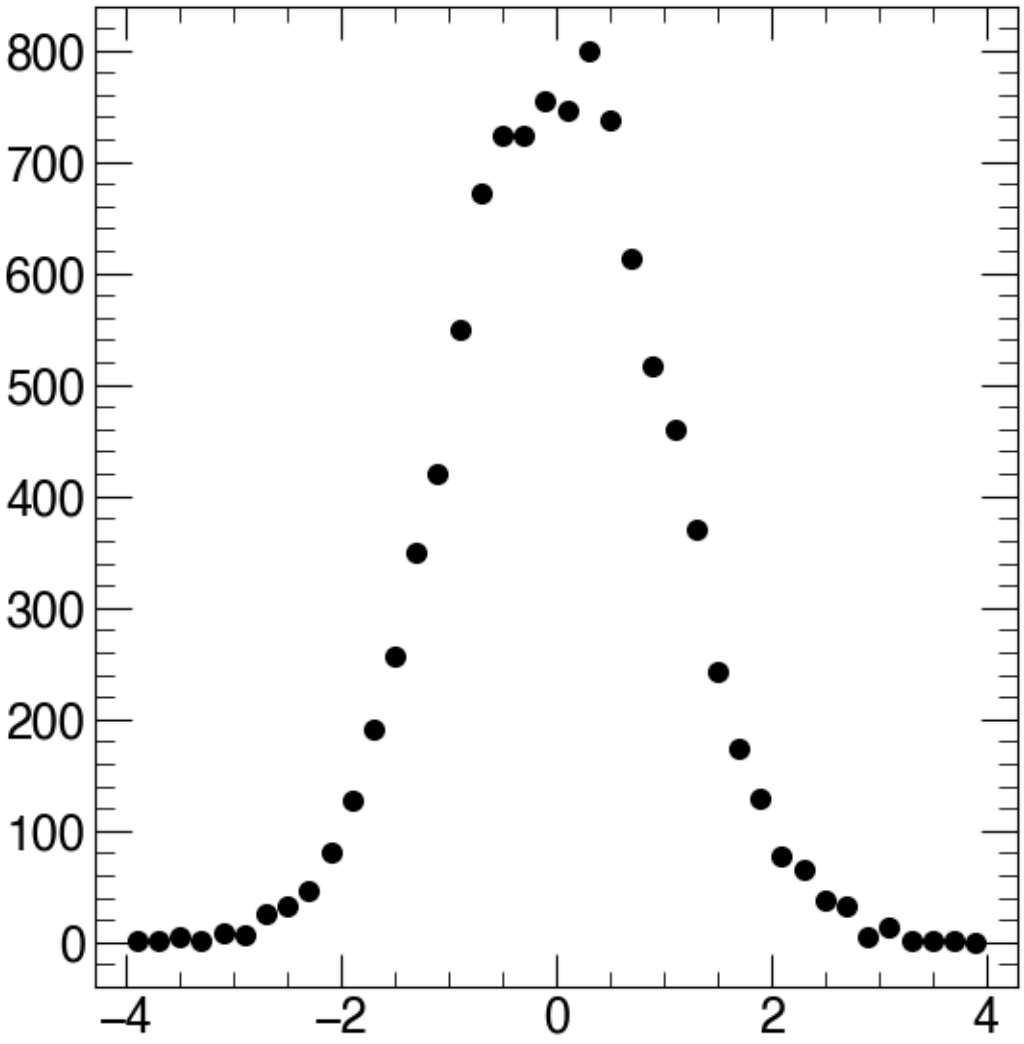


Fig. 1: Simple example

CHAPTER 5

Examples

Examples which reproduce the plots on this page (https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/PubComPlotStyle) are included in the `examples` folder. The text below goes through these in some more detail and explains what each line is doing.

5.1 1. Basic data and prediction plot

Import necessary packages

```
import numpy as np
import ROOT
import mATLASplotlib
```

Generate some ROOT data - MC prediction

```
# MC prediction
hpx_MC = ROOT.TH1F("hpx_MC", "This is the MC px distribution", 40, -4, 4)
for x in np.random.normal(size=50000):
    hpx_MC.Fill(x, 0.2)

# Pseudodata
hpx_data = ROOT.TH1F("hpx_data", "This is the data px distribution", 40, -4, 4)
for x in np.random.normal(size=10000):
    hpx_data.Fill(x)

print "Data integral, max", hpx_data.Integral(), hpx_data.GetMaximum()
```

Now we fit the data (using ROOT again)

```
fit_fn = ROOT.TF1("fit_fn", "gaus", -4, 4)
hpx_data.Fit(fit_fn, "LEMN")
mu, sigma = fit_fn.GetParameter(1), fit_fn.GetParameter(2)
mu_err, sigma_err = fit_fn.GetParError(1), fit_fn.GetParError(2)
```

Open a canvas as a context manager and plot these three datasets

```
with mATLASplotlib.canvases.Simple(shape="landscape") as canvas:
    canvas.plot_dataset(hpx_data, style="scatter yerror", label="Data 2009", colour=
↪"black")
    canvas.plot_dataset(hpx_MC, style="bar", label="Non-diffractive minimum bias",␣
↪colour="#ffff00", edgecolour="black")
    canvas.plot_dataset(fit_fn, style="smooth line", label="Gaussian fit", colour="red
↪")
```

Note that three different styles are used here: *scatter*, *bar* and *line*. The additional words in these arguments: `yerror` and `smooth` specify particular additional plotting properties - namely error bars in the y-direction and a smooth interpolation between the points.

The `label` argument to *plot_dataset* is used to build the legend which can then be drawn in one line

```
canvas.add_legend(0.04, 0.92, fontsize=16, anchor_to="upper left")
```

Now some arbitrary text can be drawn on the plot, including two special pieces of text needed for ATLAS plots - the ATLAS label and the luminosity.

```
canvas.add_text(0.04, 0.6, "$\mu = ({{{0:.2f}}}\pm{{{1:.2f}}})\,$ GeV\n$\sigma = ({{
↪{0:.2f}}}\pm{{{1:.2f}}})\,$ GeV".format(mu, mu_err, sigma, sigma_err), fontsize=16)
canvas.add_ATLAS_label(0.96, 0.92, fontsize=20, plot_type="Preliminary", anchor_to=
↪"upper right")
canvas.add_luminosity_label(0.96, 0.85, fontsize=20, sqrts_TeV=0.9, luminosity=None,␣
↪anchor_to="upper right")
```

The axis titles are set according to the data being used

```
canvas.set_axis_label("x", "$p_x$ [GeV]")
canvas.set_axis_label("y", "Events / 0.2 GeV")
```

Here the ranges are explicitly defined. Without this, the `matplotlib` defaults (which are usually good) will be used.

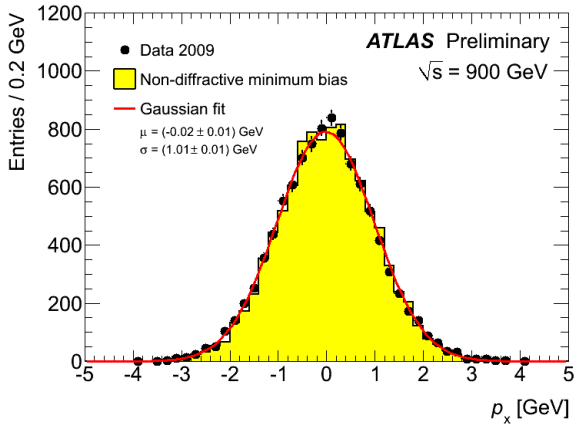
```
canvas.set_axis_range("x", (-5.0, 5.0))
canvas.set_axis_range("y", (0, 1200))
```

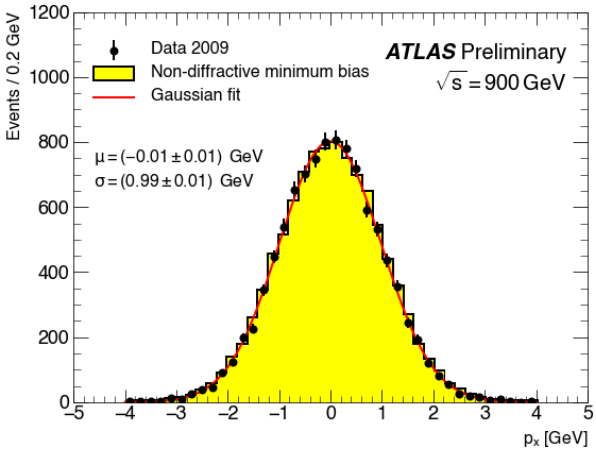
Here the default choice of axis labels on the x-axis is overridden to match the reference image

```
canvas.set_axis_ticks("x", [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5])
```

And the file is saved

```
canvas.save("example_fig_01")
```





The left-hand image is from the official ATLAS guide, the one on the right uses `mATLASplotlib`

5.2 1. More complicated - theory band and log-scale

Import necessary packages

```
from mATLASplotlib import canvases
import array
import ROOT
import numpy as np
```

Load distributions (these numbers are from the example ROOT file produced by ATLAS)

```
# Take the NLO QCD prediction from the ATLAS ROOT file
NLOQCD_x = array.array("d", [150, 250, 350, 450, 550, 650, 750, 850, 950, 1050, 1150,␣
↪1250, 1350, 1450, 1550, 1650, 1750, 1850, 1950, 2050, 2150, 2250, 2350, 2450, 2550,␣
↪2650, 2750, 2850, 2950, 3050, 3150, 3250, 3350, 3450, 3550, 3650, 3750, 3850, 3950,␣
↪4050, 4150, 4250, 4350, 4450, 4550, 4650, 4750, 4850, 4950, 5050, 5150, 5250, 5350,␣
↪5450, 5550, 5650, 5750, 5850, 5950, 6050, 6150, 6250, 6350, 6450, 6550, 6650, 6750,␣
↪6850, 6950])
NLOQCD_ex = array.array("d", [50] * len(NLOQCD_x))
NLOQCD_y = array.array("d", [6.6596e+06, 318321, 48380.3, 10957.2, 3235.47, 1232.35,␣
↪512.05, 230.488, 114.084, 60.5025, 31.1572, 17.6683, 10.3007, 6.14975, 3.71552, 2.
↪52298, 1.44746, 1.01675, 0.641863, 0.38725, 0.263351, 0.207581, 0.103852, 0.0813621,
↪ 0.0507964, 0.0374186, 0.0239413, 0.0179119, 0.0105439, 0.00724193, 0.00543263, 0.
↪00353234, 0.00237385, 0.00137003, 0.000949618, 0.000670692, 0.000455441, 0.
↪000278849, 0.000175337, 0.000121832, 8.35938e-05, 4.20491e-05, 2.53419e-05, 1.83e-
↪05, 1.[illegible], 3.[illegible], 1.[illegible], 6.[illegible], 2.47147e-06, 1.32182e-06, 5.26587e-07, 3.
↪89889e-07, 1.85564e-07, 1.05617e-07, 5.02298e-08, 2.37682e-08, 1.00941e-08, 4.
↪17607e-09, 1.74077e-09, 6.47905e-10, 2.90492e-10, 8.31421e-11, 2.45835e-11, 5.
↪51966e-12, 1.23854e-12, 2.12108e-13, 3.51307e-14, 3.83912e-15, 2.02972e-16, 1.
↪93257e-17])
```

(continues on next page)

(continued from previous page)

```
NLOQCD_eyl = array.array("d", [592438, 26408.5, 4624.47, 1045.14, 310.249, 132.484,␣
↪57.4808, 26.5156, 13.8826, 7.8785, 3.91181, 2.34453, 1.39327, 0.857228, 0.534341, 0.
↪386117, 0.211819, 0.15815, 0.101274, 0.0583714, 0.0418432, 0.0364608, 0.0162442, 0.
↪0134506, 0.00859996, 0.00647774, 0.0042077, 0.00337713, 0.00183224, 0.00130241, 0.
↪000980045, 0.000692273, 0.000480861, 0.00027158, 0.000186696, 0.000140853, 9.95418e-
↪05, 6.30202e-05, 3.98933e-05, 2.87898e-05, 2.16196e-05, 1.04431e-05, 6.53897e-06, 5.
↪1248e-06, 3.04996e-06, 1.71619e-06, 1.10496e-06, 8.44941e-07, 4.69963e-07, 1.85741e-
↪07, 1.51345e-07, 7.70224e-08, 4.67813e-08, 2.32868e-08, 1.17352e-08, 5.41192e-09, 2.
↪35221e-09, 1.05666e-09, 4.15396e-10, 2.01271e-10, 6.30795e-11, 1.97053e-11, 4.
↪85011e-12, 1.15449e-12, 2.10144e-13, 3.78813e-14, 4.13635e-15, 2.06829e-16, 1.
↪87872e-17])
NLOQCD_eyh = array.array("d", [476290, 17188.1, 3472.85, 756.231, 234.15, 115.386, 53.
↪9145, 26.1371, 14.5604, 8.67245, 4.40793, 2.79054, 1.68753, 1.07432, 0.691266, 0.
↪496964, 0.292805, 0.212455, 0.140897, 0.0844782, 0.0623834, 0.0518593, 0.0270745, 0.
↪0203713, 0.0140546, 0.0102316, 0.0069444, 0.00551092, 0.0030732, 0.00229263, 0.
↪00169067, 0.00117577, 0.000835395, 0.000520959, 0.000336736, 0.000257673, 0.
↪00017855, 0.000120173, 7.67702e-05, 5.03502e-05, 3.96392e-05, 2.10842e-05, 1.32891e-
↪05, 9.44563e-06, 5.76911e-06, 3.43875e-06, 2.11638e-06, 1.48622e-06, 7.97241e-07, 4.
↪54554e-07, 2.77689e-07, 1.50687e-07, 8.06223e-08, 4.46708e-08, 2.25211e-08, 1.1174e-
↪08, 4.73384e-09, 2.2669e-09, 9.73032e-10, 4.7206e-10, 1.81057e-10, 7.06126e-11, 2.
↪27797e-11, 7.87034e-12, 2.46757e-12, 7.2124e-13, 1.37511e-13, 1.14176e-14, 1.02065e-
↪15])
g_NLOQCD = ROOT.TGraphAsymmErrors(len(NLOQCD_x), NLOQCD_x, NLOQCD_y, NLOQCD_ex,␣
↪NLOQCD_ex, NLOQCD_eyl, NLOQCD_eyh)
```

Generate some data which looks like the QCD background plus a small amount of signal

```
# Generate some ROOT data based on the NLO QCD prediction
h_data = ROOT.TH1F("hpx_data", "This is the data px distribution", 69, 100, 7000)
r1 = 0.4 * (np.random.uniform(size=len(NLOQCD_x)) + 2)
r2 = 0.4 * (np.random.uniform(size=len(NLOQCD_x)) + 2)
for x, y, _r1, _r2 in zip(NLOQCD_x, NLOQCD_y, r1, r2):
    data_y = _r1 * y + _r1 *_r2**2 * x / 50000.
    h_data.SetBinContent(h_data.FindFixBin(x), data_y)
    h_data.SetBinError(h_data.FindFixBin(x), np.sqrt(data_y * 1000) / 200.)
```

Open a canvas and plot the datasets

```
with mATLASplotlib.canvases.Simple(shape="square") as canvas:
    canvas.plot_dataset(h_data, style="scatter yerror", label="Data 2009", colour=
↪"black")
    canvas.plot_dataset(g_NLOQCD, style="binned band central line", label="NLO QCD",␣
↪colour="#ffff00", line_colour="black")
```

next the legend and text

```
canvas.add_legend(0.45, 0.75, fontsize=20, anchor_to="upper left")
canvas.add_luminosity_label(0.15, 0.9, fontsize=20, sqrts_TeV=14, luminosity=None,␣
↪anchor_to="upper left")
canvas.add_text(0.53, 0.9, r"$|\eta_{jet}| < 0.5$", fontsize=20, anchor_to="upper left
↪")
canvas.add_ATLAS_label(0.05, 0.05, fontsize=20, plot_type="Preliminary", anchor_to=
↪"lower left")
```

and the axis titles and ranges.

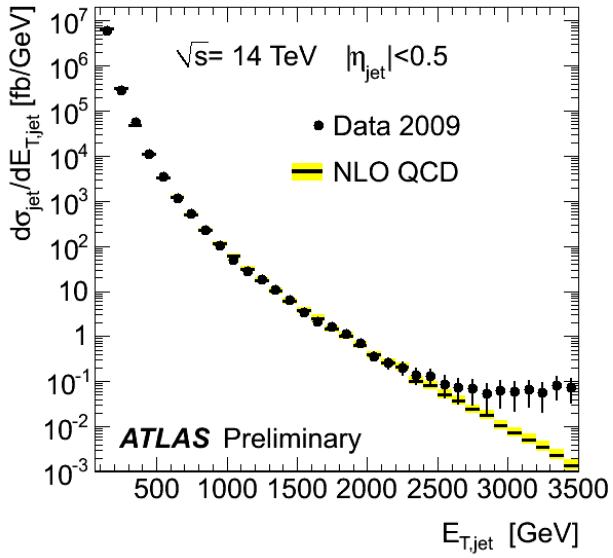
```
canvas.set_axis_label("x", r"$E_{T,jet}$  [GeV]")
canvas.set_axis_label("y", r"$d\sigma_{jet}/dE_{T,jet}$ [fb/GeV]")
canvas.set_axis_range("x", (60.0, 3500.0))
canvas.set_axis_range("y", (1e-3, 2e7))
```

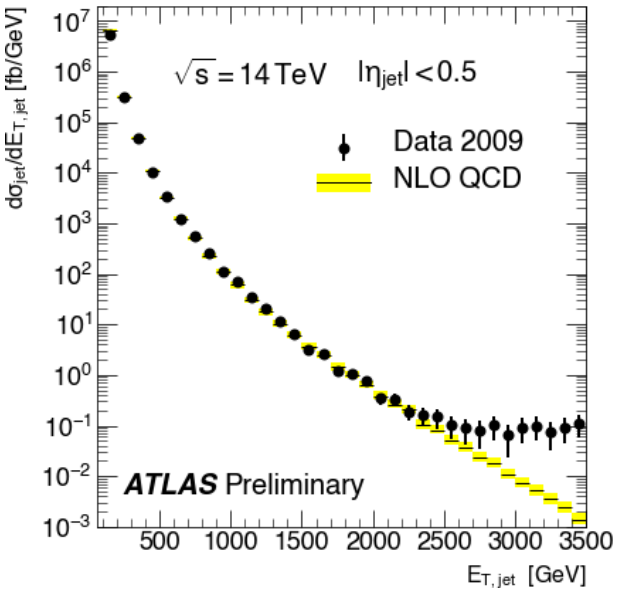
For this plot we want the y-axis to be on a log-scale and to specify the ticks

```
canvas.set_axis_log("y")
canvas.set_axis_ticks("x", [500, 1000, 1500, 2000, 2500, 3000, 3500])
canvas.set_axis_ticks("y", [1e-3, 1e-2, 1e-1, 1, 10, 1e2, 1e3, 1e4, 1e5, 1e6, 1e7])
```

Finally save it

```
canvas.save("example_fig_02", extension="png")
```





As before, the left-hand image is from the official ATLAS guide, the one on the right uses `mATLASplotlib`

CHAPTER 6

API

The API for mATLASplotlib can be seen in the following links

6.1 canvases

6.1.1 base_canvas

This module provides the BaseCanvas canvas.

class mATLASplotlib.canvases.base_canvas.**BaseCanvas**(*shape='square'*, ***kwargs*)

Bases: object

Base class for canvas properties.

__init__(*shape='square'*, ***kwargs*)

Set up universal canvas properties.

Parameters **shape** (*str*) – use either the 'square', 'landscape' or 'portrait' ATLAS proportions

Keyword Arguments

- **log_type** (*str*) – set 'x', 'y' or both ('xy') axes to log-scale
- **x_ticks_extra** (*list*) – list of additional minor ticks to label (only used when the x-axis is on a log scale)
- **x_tick_labels** (*iterable*) – list of tick labels for the x-axis
- **x_tick_label_size** (*float*) – fontsize for x-axis tick labels
- **y_tick_labels** (*iterable*) – list of tick labels for the y-axis
- **y_tick_label_size** (*float*) – fontsize for y-axis tick labels

add_ATLAS_label(*x*, *y*, *plot_type=None*, *anchor_to='lower left'*, *fontsize=None*, *axes=None*)

Add an ATLAS label to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **plot_type** (*str*) – Preliminary/Internal/Work-In-Progress/None
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_legend(*x*, *y*, *anchor_to='lower left'*, *fontsize=None*, *axes=None*)
Add a legend to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_luminosity_label(*x*, *y*, *sqrts_TeV*, *luminosity*, *units='fb-1'*, *anchor_to='lower left'*, *fontsize=14*, *axes=None*)
Add a luminosity label to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **sqrts_TeV** (*float*) – centre-of-mass energy in TeV.
- **luminosity** (*float*) – luminosity.
- **units** (*str*) – luminosity units.
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_text(*x*, *y*, *text*, ***kwargs*)
Add text to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **text** (*str*) – text to add.

auto_tick_intervals = [0.001, 0.002, 0.0025, 0.004, 0.005, 0.01, 0.02, 0.025, 0.04, 0.
List of sensible tick intervals

close()
Close the figure to free up memory. Not needed when this object is used as a context manager.

get_axis_label(*axis_name*)
Get the label for the chosen axis

Parameters **axis_name** (*str*) – which axis to use.

Returns axis label

Return type str

get_axis_range(*axis_name*)
Get the range for the chosen axis

Parameters **axis_name** (*str*) – which axis to use.

Returns axis range

Return type tuple

location_map = {'centre left': ['left', 'center'], 'centre right': ['right', 'center
Map of locations to matplotlib coordinates

plot_dataset(**args*, ***kwargs*)
Plot a dataset. Non-keyword arguments will be interpreted as the dataset to be plotted. Keyword arguments will be interpreted as style arguments.

Positional Arguments

- **args**: (*ROOT.TObject*, *iterable*, *numpy array*) – plottable information which is passed to *Dataset* to be interpreted

Keyword Arguments

- **remove_zeros**: (*bool*) – prune any points in the dataset for which the y-value is 0
- **axes**: (*str*) – which axes to use (defaults to the main subplot)
- **style**: (*str*) – which of the plotters in *plotters* to use
- **label**: (*str*) – label to use in automatic legend generation
- **sort_as**: (*str*) – override

save(*output_name*, *extension='pdf'*)
Save the current state of the canvas to a file.

Parameters

- **output_name** (*str*) – name of output file.
- **extension** (*str or list*) – type of output to produce.

set_axis_label(*axis_name*, *axis_label*, *fontsize=16*)
Set the maximum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **axis_label** (*str*) – desired label.
- **fontsize** (*float*) – font size to use

set_axis_log(*axis_names*)
Set the specified axis to be on a log-scale.

Parameters **axis_names** (*str*) – which axis (or axes) to apply this to.

set_axis_max(*axis_name*, *maximum*)
Set the maximum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **maximum** (*float*) – desired axis maximum.

set_axis_min(*axis_name*, *minimum*)
Set the minimum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **minimum** (*float*) – desired axis minimum.

set_axis_range(*axis_name*, *axis_range*)
Set the range for the given axis.

Parameters

- **axis_name** (*str.*) – which axis to apply this to.
- **axis_range** (*iterable*) – desired axis range.

set_axis_tick_ndp(*axis_name*, *ndp*)
Set number of decimal places to show.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **ndp** (*int*) – how many decimal places to show.

set_axis_ticks(*axis_name*, *ticks*)
Set the position of the axis ticks.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **ticks** (*iterable*) – desired tick positions.

set_title(*title*)
Set the figure title.

Parameters **title** (*str*) – figure title to use

x_tick_label_size
Label size for x-ticks

x_tick_labels
Labels for x-ticks

y_tick_label_size
Label size for y-ticks

y_tick_labels
Labels for y-ticks

6.1.2 panelled

This module provides the `Panelled` canvas.

class mATLASplotlib.canvases.panelled.**Panelled**(*shape='portrait'*, *n_panels=3*, *top_panel_fraction=0.16*, ***kwargs*)
Bases: *mATLASplotlib.canvases.base_canvas.BaseCanvas*

Panelled canvas with standard ATLAS setup.

__init__(*shape='portrait'*, *n_panels=3*, *top_panel_fraction=0.16*, ***kwargs*)
Set up Panelled canvas properties.

The canvas consists of a single top panel and `n_panels` equally sized additional panels underneath. These additional panels are called `plot0`, `plot1`, etc. with the numbering starting from the top.

Parameters

- **shape** (*str*) – use either the 'square', 'landscape' or 'portrait' ATLAS proportions
- **n_panels** (*int*) – how many panels to include
- **top_panel_fraction** (*float*) – fraction of vertical space that the top panel should use up

Keyword Arguments as for *BaseCanvas*

add_ATLAS_label(*x*, *y*, *plot_type=None*, *anchor_to='lower left'*, *fontsize=None*, *axes=None*)
Add an ATLAS label to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **plot_type** (*str*) – Preliminary/Internal/Work-In-Progress/None
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_legend(*x*, *y*, *anchor_to='lower left'*, *fontsize=None*, *axes=None*)
Add a legend to the canvas at (x, y).

If added to the `top` panel then all elements from the lower panels will be included in it.

Arguments as for *BaseCanvas.add_legend()*

add_luminosity_label(*x*, *y*, *sqrts_TeV*, *luminosity*, *units='fb-1'*, *anchor_to='lower left'*, *fontsize=14*, *axes=None*)
Add a luminosity label to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **sqrts_TeV** (*float*) – centre-of-mass energy in TeV.
- **luminosity** (*float*) – luminosity.
- **units** (*str*) – luminosity units.
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_text(*x*, *y*, *text*, ***kwargs*)
Add text to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).

- **y** (*float*) – y position (as a fraction of the canvas height).
- **text** (*str*) – text to add.

auto_tick_intervals = [0.001, 0.002, 0.0025, 0.004, 0.005, 0.01, 0.02, 0.025, 0.04, 0.

bottom_panel
Name of the bottom-most panel.

close()
Close the figure to free up memory. Not needed when this object is used as a context manager.

get_axis_label(*axis_name*)
Get the label for the chosen axis

Parameters **axis_name** (*str*) – which axis to use.

Returns axis label

Return type str

get_axis_range(*axis_name*)
Get the range for the chosen axis

Parameters **axis_name** (*str*) – which axis to use.

Returns axis range

Return type tuple

location_map = {'centre left': ['left', 'center'], 'centre right': ['right', 'center

plot_dataset(**args*, ***kwargs*)
Plot a dataset. Non-keyword arguments will be interpreted as the dataset to be plotted. Keyword arguments will be interpreted as style arguments.

Positional Arguments

- **args**: (*ROOT.TObject*, *iterable*, *numpy array*) – plottable information which is passed to *Dataset* to be interpreted

Keyword Arguments

- **remove_zeros**: (*bool*) – prune any points in the dataset for which the y-value is 0
- **axes**: (*str*) – which axes to use (defaults to the main subplot)
- **style**: (*str*) – which of the plotters in *plotters* to use
- **label**: (*str*) – label to use in automatic legend generation
- **sort_as**: (*str*) – override

save(*output_name*, *extension='pdf'*)
Save the current state of the canvas to a file.

Parameters

- **output_name** (*str*) – name of output file.
- **extension** (*str or list*) – type of output to produce.

set_axis_label(*axis_name*, *axis_label*, *fontsize=16*)
Set the maximum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.

- **axis_label** (*str*) – desired label.
- **fontsize** (*float*) – font size to use

set_axis_log(*axis_names*)
Set the specified axis to be on a log-scale.

Parameters **axis_names** (*str*) – which axis (or axes) to apply this to.

set_axis_max(*axis_name*, *maximum*)
Set the maximum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **maximum** (*float*) – desired axis maximum.

set_axis_min(*axis_name*, *minimum*)
Set the minimum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **minimum** (*float*) – desired axis minimum.

set_axis_range(*axis_name*, *axis_range*)
Set the range for the given axis.

Parameters

- **axis_name** (*str.*) – which axis to apply this to.
- **axis_range** (*iterable*) – desired axis range.

set_axis_tick_ndp(*axis_name*, *ndp*)
Set number of decimal places to show.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **ndp** (*int*) – how many decimal places to show.

set_axis_ticks(*axis_name*, *ticks*)
Set the position of the axis ticks.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **ticks** (*iterable*) – desired tick positions.

set_title(*title*)
Set the figure title.

Parameters **title** (*str*) – figure title to use

x_tick_label_size
Label size for x-ticks

x_tick_labels
Labels for x-ticks

y_tick_label_size
Label size for y-ticks

y_tick_labels
Labels for y-ticks

6.1.3 ratio

This module provides the `Ratio` canvas.

class mATLASplotlib.canvases.ratio.**Ratio**(*shape='square'*, *line_ypos=1.0*, ***kwargs*)
Bases: *mATLASplotlib.canvases.base_canvas.BaseCanvas*

Ratio canvas with standard ATLAS setup

__init__(*shape='square'*, *line_ypos=1.0*, ***kwargs*)
Set up Ratio canvas properties.

Parameters

- **shape** (*str*) – use either the 'square', 'landscape' or 'portrait' ATLAS proportions
- **line_ypos** (*float*) – where to draw the reference line in the ratio plot

Keyword Arguments as for *BaseCanvas*

add_ATLAS_label(*x*, *y*, *plot_type=None*, *anchor_to='lower left'*, *fontsize=None*, *axes=None*)
Add an ATLAS label to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **plot_type** (*str*) – Preliminary/Internal/Work-In-Progress/None
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_legend(*x*, *y*, *anchor_to='lower left'*, *fontsize=None*, *axes=None*)
Add a legend to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_luminosity_label(*x*, *y*, *sqrts_TeV*, *luminosity*, *units='fb-1'*, *anchor_to='lower left'*, *fontsize=14*, *axes=None*)
Add a luminosity label to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **sqrts_TeV** (*float*) – centre-of-mass energy in TeV.

- **luminosity** (*float*) – luminosity.
- **units** (*str*) – luminosity units.
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_text(*x*, *y*, *text*, ***kwargs*)
Add text to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **text** (*str*) – text to add.

auto_tick_intervals = [0.001, 0.002, 0.0025, 0.004, 0.005, 0.01, 0.02, 0.025, 0.04, 0.

close()
Close the figure to free up memory. Not needed when this object is used as a context manager.

get_axis_label(*axis_name*)
Get the label for the chosen axis

Parameters **axis_name** (*str*) – which axis to use.

Returns axis label

Return type str

get_axis_range(*axis_name*)
Get the range for the chosen axis

Parameters **axis_name** (*str*) – which axis to use.

Returns axis range

Return type tuple

location_map = {'centre left': ['left', 'center'], 'centre right': ['right', 'center

plot_dataset(**args*, ***kwargs*)
Plot a dataset. Non-keyword arguments will be interpreted as the dataset to be plotted. Keyword arguments will be interpreted as style arguments.

Positional Arguments

- **args**: (*ROOT.TObject*, *iterable*, *numpy array*) – plottable information which is passed to *Dataset* to be interpreted

Keyword Arguments

- **remove_zeros**: (*bool*) – prune any points in the dataset for which the y-value is 0
- **axes**: (*str*) – which axes to use (defaults to the main subplot)
- **style**: (*str*) – which of the plotters in *plotters* to use
- **label**: (*str*) – label to use in automatic legend generation
- **sort_as**: (*str*) – override

save(*output_name*, *extension='pdf'*)
Save the current state of the canvas to a file.

Parameters

- **output_name** (*str*) – name of output file.
- **extension** (*str or list*) – type of output to produce.

set_axis_label(*axis_name*, *axis_label*, *fontsize=16*)
Set the maximum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **axis_label** (*str*) – desired label.
- **fontsize** (*float*) – font size to use

set_axis_log(*axis_names*)
Set the specified axis to be on a log-scale.

Parameters **axis_names** (*str*) – which axis (or axes) to apply this to.

set_axis_max(*axis_name*, *maximum*)
Set the maximum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **maximum** (*float*) – desired axis maximum.

set_axis_min(*axis_name*, *minimum*)
Set the minimum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **minimum** (*float*) – desired axis minimum.

set_axis_range(*axis_name*, *axis_range*)
Set the range for the given axis.

Parameters

- **axis_name** (*str.*) – which axis to apply this to.
- **axis_range** (*iterable*) – desired axis range.

set_axis_tick_ndp(*axis_name*, *ndp*)
Set number of decimal places to show.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **ndp** (*int*) – how many decimal places to show.

set_axis_ticks(*axis_name*, *ticks*)
Set the position of the axis ticks.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **ticks** (*iterable*) – desired tick positions.

set_title(*title*)
Set the figure title.

Parameters **title** (*str*) – figure title to use

x_tick_label_size
Label size for x-ticks

x_tick_labels
Labels for x-ticks

y_tick_label_size
Label size for y-ticks

y_tick_labels
Labels for y-ticks

6.1.4 simple

This module provides the Simple canvas.

class mATLASplotlib.canvases.simple.**Simple**(*shape='square'*, ***kwargs*)
Bases: *mATLASplotlib.canvases.base_canvas.BaseCanvas*

Simple canvas with standard ATLAS setup.

__init__(*shape='square'*, ***kwargs*)
Set up universal canvas properties.

Parameters **shape** (*str*) – use either the 'square', 'landscape' or 'portrait' ATLAS proportions

Keyword Arguments

- **log_type** (*str*) – set 'x', 'y' or both ('xy') axes to log-scale
- **x_ticks_extra** (*list*) – list of additional minor ticks to label (only used when the x-axis is on a log scale)
- **x_tick_labels** (*iterable*) – list of tick labels for the x-axis
- **x_tick_label_size** (*float*) – fontsize for x-axis tick labels
- **y_tick_labels** (*iterable*) – list of tick labels for the y-axis
- **y_tick_label_size** (*float*) – fontsize for y-axis tick labels

add_ATLAS_label(*x*, *y*, *plot_type=None*, *anchor_to='lower left'*, *fontsize=None*, *axes=None*)
Add an ATLAS label to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **plot_type** (*str*) – Preliminary/Internal/Work-In-Progress/None
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_legend(*x*, *y*, *anchor_to='lower left'*, *fontsize=None*, *axes=None*)
Add a legend to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_luminosity_label(*x*, *y*, *sqrts_TeV*, *luminosity*, *units='fb-1'*, *anchor_to='lower left'*, *fontsize=14*, *axes=None*)

Add a luminosity label to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **sqrts_TeV** (*float*) – centre-of-mass energy in TeV.
- **luminosity** (*float*) – luminosity.
- **units** (*str*) – luminosity units.
- **anchor_to** (*str*) – anchor point (one of the options in location_map).
- **fontsize** (*float*) – font size.
- **axes** (*str*) – which of the different axes in this canvas to use.

add_text(*x*, *y*, *text*, ***kwargs*)

Add text to the canvas at (x, y).

Parameters

- **x** (*float*) – x position (as a fraction of the canvas width).
- **y** (*float*) – y position (as a fraction of the canvas height).
- **text** (*str*) – text to add.

auto_tick_intervals = [0.001, 0.002, 0.0025, 0.004, 0.005, 0.01, 0.02, 0.025, 0.04, 0.

close()

Close the figure to free up memory. Not needed when this object is used as a context manager.

get_axis_label(*axis_name*)

Get the label for the chosen axis

Parameters **axis_name** (*str*) – which axis to use.

Returns axis label

Return type str

get_axis_range(*axis_name*)

Get the range for the chosen axis

Parameters **axis_name** (*str*) – which axis to use.

Returns axis range

Return type tuple

location_map = {'centre left': ['left', 'center'], 'centre right': ['right', 'center

plot_dataset(**args*, ***kwargs*)
Plot a dataset. Non-keyword arguments will be interpreted as the dataset to be plotted. Keyword arguments will be interpreted as style arguments.

Positional Arguments

- **args**: (*ROOT.TObject*, *iterable*, *numpy array*) – plottable information which is passed to *Dataset* to be interpreted

Keyword Arguments

- **remove_zeros**: (*bool*) – prune any points in the dataset for which the y-value is 0
- **axes**: (*str*) – which axes to use (defaults to the main subplot)
- **style**: (*str*) – which of the plotters in *plotters* to use
- **label**: (*str*) – label to use in automatic legend generation
- **sort_as**: (*str*) – override

save(*output_name*, *extension='pdf'*)
Save the current state of the canvas to a file.

Parameters

- **output_name** (*str*) – name of output file.
- **extension** (*str or list*) – type of output to produce.

set_axis_label(*axis_name*, *axis_label*, *fontsize=16*)
Set the maximum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **axis_label** (*str*) – desired label.
- **fontsize** (*float*) – font size to use

set_axis_log(*axis_names*)
Set the specified axis to be on a log-scale.

Parameters **axis_names** (*str*) – which axis (or axes) to apply this to.

set_axis_max(*axis_name*, *maximum*)
Set the maximum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **maximum** (*float*) – desired axis maximum.

set_axis_min(*axis_name*, *minimum*)
Set the minimum value for the given axis.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **minimum** (*float*) – desired axis minimum.

set_axis_range(*axis_name*, *axis_range*)
Set the range for the given axis.

Parameters

- **axis_name** (*str.*) – which axis to apply this to.
- **axis_range** (*iterable*) – desired axis range.

set_axis_tick_ndp(*axis_name*, *ndp*)
Set number of decimal places to show.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **ndp** (*int*) – how many decimal places to show.

set_axis_ticks(*axis_name*, *ticks*)
Set the position of the axis ticks.

Parameters

- **axis_name** (*str*) – which axis to apply this to.
- **ticks** (*iterable*) – desired tick positions.

set_title(*title*)
Set the figure title.

Parameters **title** (*str*) – figure title to use

x_tick_label_size
Label size for x-ticks

x_tick_labels
Labels for x-ticks

y_tick_label_size
Label size for y-ticks

y_tick_labels
Labels for y-ticks

6.2 converters

6.2.1 dataset

This module provides the `Dataset` class.

class mATLASplotlib.converters.dataset.**Dataset**(**args*, ***kwargs*)
Bases: `object`

Container for plottable datasets.

__init__(**args*, ***kwargs*)
Constructor - specify values and error pair separately for each dimension.

Arguments will be interpreted as the dataset to be plotted.

Examples

- x vs y : __init__([1,2,3], [4,9,16])
- x vs y with y_errors : __init__([1,2,3], None, [4,9,16], [2,3,4])
- z values at each (x, y) point : __init__([1, 2], [3, 4], [3, 4, 6, 8])

Positional Arguments

- **args**: (*ROOT.TObject*, *iterable*, *numpy array*) – plottable information which is used to build a `Dataset`

Keyword Arguments

- **x_values**: (*iterable*) – list of points along the x-axis
- **x_error_pairs**: (*iterable*) – list of error pairs along the x-axis
- **y_values**: (*iterable*) – list of points along the y-axis
- **y_error_pairs**: (*iterable*) – list of error pairs along the y-axis
- **z_values**: (*iterable*) – list of points along the z-axis
- **z_error_pairs**: (*iterable*) – list of error pairs along the z-axis

Raises

- **`AssertionError`** – arguments are not correctly sized
- **`ValueError`** – arguments cannot be interpreted

`construct_2D_bin_list`(*axes='xy'*)
Construct full set of bins when treating x and y as two sides of a 2D plot.

Returns array of x and y bin centers that cover every (x, y) combination

Return type [np.array, np.array]

Raises **`ValueError`** – unsupported axes argument

`get_dimensions`()
Get a list of dimension names.

Returns list of dimension names

Return type list(str)

6.2.2 root2data

This module provides the `root2data` class.

class `mATLASplotlib.converters.root2data.`**`root2data`**(*root_object*, *remove_zeros=False*)

Bases: `object`

Interpreter for ROOT objects.

`__init__`(*root_object*, *remove_zeros=False*)
Extract x/y/z information from a ROOT object.

Parameters

- **`root_object`** (*`ROOT.TObject`*) – ROOT input to interpret
- **`remove_zeros`** (*`bool`*) – whether to remove points with a value of 0

`construct_from_TF1`(*input_TF1*)
Read TF1 into x, y dimensions.

Parameters **`input_TF1`** (*`ROOT.TF1`*) – input TF1

`construct_from_TGraph`(*input_TGraph*)
Read TGraph into x, y dimensions.

Parameters **`input_TGraph`** (*`ROOT.TGraph`*) – input TGraph

construct_from_TGraphAsymmErrors(*input_TGraphAsymmErrors*)
Read TGraphErrors into x, y dimensions.

Parameters **input_TGraphAsymmErrors** (*ROOT.TGraphAsymmErrors*) – input TGraphAsymmErrors

construct_from_TGraphErrors(*input_TGraphErrors*)
Read TGraphErrors into x, y dimensions.

Parameters **input_TGraphErrors** (*ROOT.TGraphErrors*) – input TGraphErrors

construct_from_TH1(*input_TH1*)
Read TH1 into x, y dimensions.

Parameters **input_TH1** (*ROOT.TH1*) – input TH1

construct_from_TH2(*input_TH2*)
Read TH2 into x, y, z dimensions.

Parameters **input_TH2** (*ROOT.TH2*) – input TH2

do_zero_removal()
Remove points with zero y-value.

static valid_input(*test_object*)
Check that the input object is a valid ROOT TObject.

Parameters **test_object** (*object*) – input object to invesigate

Returns whether the input object is a ROOT TObject

Return type bool

6.3 decorations

6.3.1 atlas_text

This module provides the `draw_ATLAS_text` convenience function.

mATLASplotlib.decorations.atlas_text.**draw_ATLAS_text**(*axes*, *loc*, *align*, *plot_type=None*, *fontsize=17*)
Draw ATLAS text on axes.

Parameters

- **axes** (*str*) – axes to plot on
- **loc** (*tuple(float)*) – x and y position of text
- **align** (*tuple(str)*) – horizontal and vertical alignment of text
- **plot_type** (*str*) – Internal/Preliminary/Work-In-Progress etc.
- **fontsize** (*float*) – fontsize of legend contents

6.3.2 legend

This module provides the `Legend` class.

class mATLASplotlib.decorations.legend.**Legend**
Bases: object

Class for controlling plot legends.

__init__()
Initialise legend ordering and default fontsize.

add_dataset(*label*, *is_stack=False*, *sort_as=None*)
Add a dataset to the legend.

Parameters

- **label** (*str*) – label that will appear in the legend
- **is_stack** (*bool*) – if this is a stack plot it needs to be stored in reverse order
- **sort_as** (*str*) – override the default first-in-first-out sorting, and sort using this text instead

plot(*x*, *y*, *axes*, *anchor_to*, *fontsize*, *use_axes=False*)
Plot the legend at (x, y) on the chosen axes.

Parameters

- **x** (*float*) – x-position of legend
- **y** (*float*) – y-position of legend
- **axes** (*str*) – axes to plot on
- **anchor_to** (*str*) – which corner to anchor the (x, y) to
- **fontsize** (*float*) – fontsize of legend contents
- **use_axes** (*list[matplotlib.axes.Axes]*) – get handles and labels from all axes in list

6.3.3 text

This module provides the draw_text convenience function.

mATLASplotlib.decorations.text.**draw_text**(*text*, *axes*, *loc*, *align*, *fontsize=16*, ***kwargs*)
Draw arbitrary text strings at (x, y) on the chosen axes.

Parameters

- **text** (*str*) – text to draw
- **axes** (*str*) – axes to plot on
- **loc** (*tuple(float)*) – x and y position of text
- **align** (*tuple(str)*) – horizontal and vertical alignment of text
- **fontsize** (*float*) – fontsize of legend contents

Keyword Arguments

- **colour** (*str*) – set text colour
- **transform** (*str*) – use data or axes coordinates

6.4 plotters

6.4.1 base_plotter

This module provides the `BasePlotter` class.

class mATLASplotlib.plotters.base_plotter.**BasePlotter**(*plot_style*)
Bases: `object`

Base class for plot formatting.

__init__(*plot_style*)
Initialise common plotting properties.

Parameters **plot_style** (*str*) – which plotting style to use. Consists of plot style name, plus any additional options available in the child class.

add_to_axes(*axes*, *dataset*, ***kwargs*)
Add the chosen dataset to the chosen axes. Delegated to child classes.

Parameters

- **axes** (*matplotlib.axes*) – which axes to plot this dataset on
- **dataset** (`Dataset`) – dataset to plot

6.4.2 bar

This module provides the `BarChart` class.

class mATLASplotlib.plotters.bar_chart.**BarChart**(*plot_style*)
Bases: *mATLASplotlib.plotters.base_plotter.BasePlotter*

Plot bar chart.

add_to_axes(*axes*, *dataset*, ***kwargs*)
Add the chosen dataset to the chosen axes.

Parameters

- **axes** (*matplotlib.axes*) – which axes to plot this dataset on
- **dataset** (*matplotlib.axes*) – which axes to plot this dataset on

Keyword Arguments

- **colour** (*str*) – which face colour to use
- **edgecolour** (*float*) – which colour to use for the outline
- **edgewidth** (*float*) – how large an outline width to use
- **label** (*str*) – label to use when this appears in a legend

6.4.3 binned_band

This module provides the `BinnedBand` class.

class mATLASplotlib.plotters.binned_band.**BinnedBand**(*plot_style*)
Bases: *mATLASplotlib.plotters.base_plotter.BasePlotter*

Plot as binned band in the x-y plane.

Additional plot_style options

- *central line* – also draw a horizontal line at the vertical centre of each bin
- *central line stepped* – as for *central line* but also join these vertically at the edge of each bin

add_to_axes(*axes*, *dataset*, ***kwargs*)
Add the chosen dataset to the chosen axes.

Parameters

- **axes** (*matplotlib.axes*) – which axes to plot this dataset on
- **dataset** (*matplotlib.axes*) – which axes to plot this dataset on

Keyword Arguments

- **alpha** (*float*) – set alpha transparency
- **colour** (*str*) – which face colour to use
- **hatch** (*str*) – set hatch pattern
- **hatchcolour** (*str*) – which hatch colour to use
- **label** (*str*) – label to use when this appears in a legend
- **linecolour** (*float*) – which colour to use for the main line and for hatches
- **linewidth** (*str*) – how wide to draw the line

6.4.4 coloured_2D

This module provides the `Coloured2D` class.

class mATLASplotlib.plotters.coloured_2D.**Coloured2D**(*plot_style*)
Bases: *mATLASplotlib.plotters.base_plotter.BasePlotter*

Plot as points in the x-y plane

add_to_axes(*axes*, *dataset*, ***kwargs*)
Add the chosen dataset to the chosen axes.

Parameters

- **axes** (*matplotlib.axes*) – which axes to plot this dataset on
- **dataset** (*matplotlib.axes*) – which axes to plot this dataset on

Keyword Arguments

- **colour_map** (*str*) – which colour map to use
- **with_key** (*bool*) – draw the key (True by default)

6.4.5 get_plotter

This module provides the `get_plotter()` convenience function.

mATLASplotlib.plotters.get_plotter.**get_plotter**(*plot_style*)
Convert a plot style argument into a concrete class inheriting from BasePlotter.

Parameters **plot_style** (*str*) – which plot style to use.

Returns instantiated plotter object

Return type *BasePlotter*

Raises

- **ValueError** – No plot style provided.
- **NotImplementedError** – Unsupported plot style provided.

6.4.6 line

This module provides the `Line` class.

class mATLASplotlib.plotters.line.**Line**(*plot_style*)
Bases: *mATLASplotlib.plotters.base_plotter.BasePlotter*

Plot as a line in the x-y plane

Additional plot_style options

- *join centres* – join the centres with straight line segments
- *smooth* – draw a smooth line through the points
- *stepped* – draw a stepped line graph

add_to_axes(*axes*, *dataset*, ***kwargs*)
Add the chosen dataset to the chosen axes.

Parameters

- **axes** (*matplotlib.axes*) – which axes to plot this dataset on
- **dataset** (*matplotlib.axes*) – which axes to plot this dataset on

Keyword Arguments

- **colour** (*str*) – which face colour to use
- **label** (*str*) – label to use when this appears in a legend
- **linestyle** (*str*) – which style (dotted/dashed/solid etc.) to draw the line with
- **linewidth** (*str*) – how wide to draw the line
- **marker** (*str*) – which marker to use

6.4.7 scatter

This module provides the `Scatter` class.

class mATLASplotlib.plotters.scatter.**Scatter**(*plot_style*)
Bases: *mATLASplotlib.plotters.base_plotter.BasePlotter*

Plot as scattered points in the x-y plane

__init__(*plot_style*)
Initialise plotting properties.

Parameters **plot_style** (*str*) – which plotting style to use.

Plot_style options

- *scatter join centres* – also draw a line joining the bin centre
- *scatter xerror* – also draw error bars in the x-direction

- *scatter yerror* – also draw error bars in the y-direction

add_to_axes(*axes*, *dataset*, ***kwargs*)
Add the chosen dataset to the chosen axes.

Parameters

- **axes** (*matplotlib.axes*) – which axes to plot this dataset on
- **dataset** (*matplotlib.axes*) – which axes to plot this dataset on

Keyword Arguments

- **colour** (*str*) – which face colour to use
- **label** (*str*) – label to use when this appears in a legend
- **linestyle** (*str*) – which style (dotted/dashed/solid etc.) to draw the line with if *join centers* is specified
- **linewidth** (*str*) – how wide to draw the line
- **marker** (*str*) – which marker to use
- **with_error_bar_caps** (*bool*) – whether to draw caps on the end of the error bars

6.4.8 stack

This module provides the `Stack` class.

class mATLASplotlib.plotters.stack.**Stack**(*plot_style*)
Bases: *mATLASplotlib.plotters.base_plotter.BasePlotter*

Plot as part of a vertically stacked histogram.

add_to_axes(*axes*, *dataset*, ***kwargs*)
Add the chosen dataset to the chosen axes.

Parameters

- **axes** (*matplotlib.axes*) – which axes to plot this dataset on
- **dataset** (*matplotlib.axes*) – which axes to plot this dataset on

Keyword Arguments

- **colour** (*str*) – which face colour to use
- **hatch** (*str*) – set hatch pattern
- **hatchcolour** (*str*) – which hatch colour to use
- **label** (*str*) – label to use when this appears in a legend
- **outlinewidth** (*str*) – how wide to draw the outline

6.5 style

6.5.1 atlas

This module provides the `set_atlas()` convenience function.

mATLASplotlib.style.atlas.**set_atlas**()
Set the plotting style to ATLAS-style and then point this function to 'None' so that it can only be called once. Called on canvas creation.

6.6 matplotlib_wrapper

This module sets the default matplotlib backend - must be done before local imports.

Python Module Index

m

Index

Symbols

A

B

C

D

G

L

M

P

R

S

V

X

Y