atlas-plots Documentation

Release 0.1

Joey Carter

Mar 18, 2021

CONTENTS:

Warning: ATLAS Plot Utils (formerly *ATLAS Plots*) is now mostly deprecated!

Check out my new project, ATLAS Plots, which uses matplotlib-like syntax and idioms to produce plots in ROOT following the standard ATLAS style guidelines.

ATLAS Plot Utils is a suite of utility functions and style settings for making pretty plots with PyROOT for the ATLAS Experiment at CERN.

Writing code to produce high energy physics plots in classic C++ ROOT is notoriously cumbersome and slow. PyROOT and **atlas-plot-utils** help make things a little easier.

CHAPTER

ONE

GETTING STARTED

1.1 Before You Begin

If you can, use Python 3. Have a look at Picking a Python Interpreter on Kenneth Reitz's Guide to Python for a few reasons why.

Also have a look at the chapter on Properly Installing Python for recommendations on how to install your Python interpreter.

If you're using a Mac, you can install Python with Homebrew:

```
$ brew install python3
```

Homebrew also installs `pip3` for you, which is an alias pointing to `pip` in your Homebrew'd version of Python 3.

Note: You may encounter issues with file permissions when installing software because Homebrew needs write access to `/usr/local/`. Since `brew install` will not work with root privileges (unless `brew` itself is owned by root), you can change the ownership of the contents of `/usr/local/`, or wherever Homebrew installs software, with,

```
$ sudo chown -R $(whoami) $(brew --prefix)/*
```

Proceed with caution, however, if you have other non-Homebrew'd software here.

1.2 Installing ROOT

ROOT is a scientific software framework developed at CERN for data analysis in high energy physics. The easiest way to install ROOT if you're using a Mac is also with Homebrew:

```
$ brew install root --with-python3 --without-python
```

This compiles ROOT from source which can take upwards of 90 minutes, but at the advantage of giving you access to the ROOT libraries from Python 3. `brew install root` on the other hand installs a pre-compiled version of ROOT, which by default only supports Python 2.

If you don't like using Homebrew, you can always download ROOT from https://root.cern.ch/downloading-root and install it manually. Or if you're feeling really adventurous, you can clone the ROOT GitHub repository and install it from source.

Note: Because ROOT depends on several installation-dependent environment variables to function properly, you should add the following commands to your shell initialization script (.bashrc/.profile/etc.), or call them directly before using ROOT.

For bash users: `. /usr/local/bin/thisroot.sh`

For zsh users: `pushd /usr/local >/dev/null; . bin/thisroot.sh; popd >/dev/null`

For csh/tcsh users: `source /usr/local/bin/thisroot.csh`

1.3 PyROOT

PyROOT is a Python extension module that allows the user to interact with any ROOT class from the Python interpreter. Using PyROOT is super easy. As an example, consider opening a TBrowser:

```
>>> import ROOT
>>> br = ROOT.TBrowser()
```

Or if you prefer, you can import ROOT classes directly:

```
>>> from ROOT import TBrowser
>>> br = TBrowser()
```

Note: As a shortcut, set an alias to `python -i -c "import ROOT"` to open a Python shell with ROOT ready to go.

To illustrate the power and simplicity of PyROOT, consider opening a ROOT file `data_file.root` containing a TTree called `data_tree`:

```
>>> import ROOT
>>> file = ROOT.TFile.Open("data_file.root")
>>> tree = file.data_tree
>>> tree.Print()
******************************************************************************
*Tree    :data_tree : Test ROOT tree                                         *
*Entries :     3524 : Total =         2104055 bytes  File  Size =     196761 *
*        :          : Tree compression factor =  10.76                       *
```

(continues on next page)

(continued from previous page)

```
*******************************************************************************
*Br    0 :eta       : eta/D                                                  *
*Entries :     3524 : Total  Size=     542582 bytes  File Size  =      49459 *
*Baskets :       18 : Basket Size=      32000 bytes  Compression=  10.96     *
*.............................................................................*
*Br    1 :phi       : phi/D                                                  *
*Entries :     3524 : Total  Size=     542582 bytes  File Size  =      49459 *
*Baskets :       18 : Basket Size=      32000 bytes  Compression=  10.96     *
*.............................................................................*
...
```

As a complete example, suppose you want to print all the values in the `eta` branch:

```
import ROOT

file = ROOT.TFile.Open("data_file.root")
tree = file.data_tree

for entry in tree:
    print(entry.eta)
```

Compare this to the equivalent C++ ROOT macro:

```
{
    TFile* file = TFile::Open("data_file.root");

    TTreeReader data_reader("data_tree", file);
    TTreeReaderValue<double> eta(data_reader, "eta");

    while (data_reader.Next()) {
        std::cout << *eta << std::endl;
    }
}
```

1.4 Installing atlasplots

atlasplots isn't in PyPI (yet) so for now it's best to clone the source and install as editable:

```
$ git clone git@github.com:joeycarter/atlas-plots.git
$ cd atlas-plots
$ pip install -e .
```

To uninstall:

```
$ pip uninstall atlasplots
```

1.5 Installing on lxplus

Note: Fellow ATLAS members: follow these instructions to set up **atlasplots** on `lxplus`. Other CERN folk, you may have to fill in some of the gaps to set up the latest versions of ROOT and Python.

If you're like me and want to use **atlasplots** on `lxplus` (to avoid copying over potentially large ROOT files to your local machine), there are a few extra steps involved to install it. As of writing these docs, the default version of ROOT installed on `lxplus` is 5.34/36 and the default version of Python is 2.6.6 (*sigh...*). First things first, let's get proper, recent releases of these two applications:

```
$ setupATLAS
$ lsetup root
```

Note: Careful here: setting up ROOT in this way might interfere with certain environment and `PATH` variables if you have an `Athena` release set up in your current shell.

Now we have at least ROOT 6.10/04 and Python 2.7.13 (it's no Python 3, but it'll do). With this sparkling new software ready to go, you will likely run into an issue if you start trying to install packages with `pip`: installing packages will normally work, but you can't import them. This is because while you can set up different versions of Python on `lxplus`, there is only the default system `pip` available. To get around this, you can install the latest version of `pip` yourself:

```
$ wget https://bootstrap.pypa.io/get-pip.py && python get-pip.py --user
```

This installs `pip` to `~/.local/bin/` (you can get rid of `get-pip.py` afterwards). Now to use `pip`, you can set the following alias:

```
$ alias pip="python ~/.local/bin/pip"
```

Then to install a package,

```
$ pip install <package> --user
```

The `--user` flag tells `pip` to install packages to `~/.local/lib/` (which is necessary since you don't have `sudo` privileges on `lxplus`). Finally, at long last, you can install **atlasplots**:

```
$ git clone git@github.com:joeycarter/atlas-plots.git
$ cd atlas-plots
$ pip install -e . --user
```

1.6 Basic Usage

Import the **atlasplots** package:

```
>>> import atlasplots
```

Set the ATLAS Style for plotting:

```
>>> from atlasplots import atlas_style as astyle
>>> astyle.SetAtlasStyle()
```

Add the "*ATLAS* Internal" label to a plot:

```
>>> from atlasplots import atlas_style as astyle
>>> astyle.ATLASLabel(0.2, 0.87, "Internal")
```

For a collection of complete examples, see the *Examples* section.

CHAPTER

TWO

ATLAS STYLE

This module contains the ATLAS Style definition for plotting, as well as methods for adding the ubiquitous *ATLAS* label and a few other deprecated methods for adding text to plots.

The ATLAS Style will automatically take care of setting up the ATLAS default plot formatting, and namely it will:

- Remove the ROOT default grey background, and set all canvas backgrounds to white (except for TLegend objects);
- Set the pad margins to decent values, so that your axis labels will not overlap on the axis figures;
- Select the ATLAS default font (Helvetica) at its default size;
- Select the default marker type (full circle, black);
- Increase the default line thicknesses (figures in paper are usually greatly reduced: this improves visibility in both articles and conference presentations);
- Avoid the display of any of the standard histogram decorations (title, statistics box, . . .);
- Put tick marks on top and right hand side of your plots.

See https://twiki.cern.ch/twiki/bin/view/AtlasProtected/PubComPlotStyle for the full documentation on style guidelines for ATLAS plots (requires CERN login).

Examples

Import the module:

```
>>> import ROOT
>>> from atlasplots import atlas_style as astyle
```

Set the ATLAS Style globally:

```
>>> astyle.SetAtlasStyle()
```

Add the "*ATLAS* Internal" label:

```
>>> astyle.ATLASLabel(0.2, 0.87, "Internal")
```

For a collection of complete examples, see the *Examples* section.

atlas_style.**ATLASLabel**(*x*, *y*, *text=None*, *color=1*)
Draw the ATLAS Label.

Parameters

- **x** (*float*) – x position in NDC coordinates

- **y** (*float*) – y position in NDC coordinates
- **text** (*string, optional*) – Text displayed next to label (the default is None)
- **color** (*TColor, optional*) – Text colour (the default is 1, i.e. black). See https://root.cern.ch/doc/master/classTColor.html

atlas_style.**ATLASLabelOld**(*x*, *y*, *preliminary=False*, *color=1*)
Draw the ATLAS Label (old version).

Note: Use *ATLASLabel()* instead.

Parameters

- **x** (*float*) – x position in NDC coordinates
- **y** (*float*) – y position in NDC coordinates
- **preliminary** (*bool, optional*) – If True, write "Preliminary" next to label (the default is False)
- **color** (*TColor, optional*) – Text colour (the default is 1, i.e. black). See https://root.cern.ch/doc/master/classTColor.html

atlas_style.**ATLASVersion**(*version*, *x=0.88*, *y=0.975*, *color=1*)
Draw the version number.

Parameters

- **version** (*string*) – Version number
- **x** (*float, optional*) – x position in NDC coordinates (the default is 0.88)
- **y** (*float, optional*) – y position in NDC coordinates (the default is 0.975)
- **color** (*TColor, optional*) – Text colour (the default is 1, i.e. black). See https://root.cern.ch/doc/master/classTColor.html

atlas_style.**AtlasStyle**()
Define the ATLAS style.

Returns The style object with ATLAS style settings

Return type TStyle

atlas_style.**SetAtlasStyle**()
Applying ATLAS style settings globally.

CHAPTER

THREE

PLOTTING UTILS

The main suite of utility functions.

utils.**DrawHistograms**(*hists*, *options=''*)
Draw the histograms.

The histograms should already have their formatting applied at this point

Parameters

- **hists** (*[TH1]*) – List of histograms
- **options** (*str, optional*) – Drawing options (the default is "")

utils.**DrawLine**(*x1*, *y1*, *x2*, *y2*, *color=1*, *width=1*, *style=1*, *alpha=1*)
Draw a line on a histogram.

See https://root.cern.ch/doc/master/classTAttLine.html for more information on line properties.

Parameters

- **x1** (*float*) – Line coordinates
- **y1** (*float*) – Line coordinates
- **x2** (*float*) – Line coordinates
- **y2** (*float*) – Line coordinates
- **color** (*int, optional*) – Line colour (the default is 1, i.e. black). See https://root.cern.ch/doc/master/classTColor.html. If you know the hex code, rgb values, etc., use `ROOT.TColor.GetColor()`
- **width** (*int, optional*) – Line width in pixels; between 1 and 10 (the default is 1)
- **style** (*int, optional*) – Line style; between 1 and 10 (the default is 1, i.e. solid line)
- **alpha** (*float, optional*) – Line transparency; between 0 and 1 (the default is 1, i.e. opaque)

utils.**DrawLineAt1**(*hist*, *color=1*, *width=1*, *style=1*, *alpha=1*)
Draw a horizontal line at y=1 on a histogram.

This is particularly useful for ratio plots.

See https://root.cern.ch/doc/master/classTAttLine.html for more information on line properties.

Parameters

- **hist** (*TH1*) – The histogram on which to draw the line

- **color** (*int, optional*) – Line colour (the default is 1, i.e. black). See https://root.cern.ch/doc/master/classTColor.html. If you know the hex code, rgb values, etc., use `ROOT.TColor.GetColor()`
- **width** (*int, optional*) – Line width in pixels; between 1 and 10 (the default is 1)
- **style** (*int, optional*) – Line style; between 1 and 10 (the default is 1, i.e. solid line)
- **alpha** (*float, optional*) – Line transparency; between 0 and 1 (the default is 1, i.e. opaque)

`utils.`**DrawText**(*x*, *y*, *text*, *color=1*, *size=0.05*)

Draw text.

Parameters

- **x** (*float*) – x position in NDC coordinates
- **y** (*float*) – y position in NDC coordinates
- **text** (*string, optional*) – The text
- **color** (*int, optional*) – Text colour (the default is 1, i.e. black). See https://root.cern.ch/doc/master/classTColor.html. If you know the hex code, rgb values, etc., use `ROOT.TColor.GetColor()`
- **size** (*float, optional*) – Text size See https://root.cern.ch/doc/master/classTLatex.html

`utils.`**FormatHistograms**(*hists*, *title=''*, *xtitle=''*, *ytitle=''*, *xtitle_offset=None*, *ytitle_offset=None*, *units=''*, *max=None*, *min=0*)

Format histograms and add axis labels.

Typically the y-axis label contains the bin width with units, for example, "Events / 10 GeV". The preferred way to get the bin width is at run time rather than computing it by hand and including it in the config file. So, if no units are specified, the function will try to parse the units from the x-axis label and apply it to the y-axis.

Parameters

- **hists** (*[TH1]*) – List of histograms
- **title** (*str, optional*) – Histogram title; typically empty (the default is "")
- **xtitle** (*str, optional*) – x-axis label (the default is "")
- **ytitle** (*str, optional*) – y-axis label (the default is "")
- **xtitle_offset** (*float, optional*) – Label offset from x-axis (the default is None, i.e. use ROOT's default)
- **ytitle_offset** (*float, optional*) – Label offset from y-axis (the default is None, i.e. use ROOT's default)
- **units** (*str, optional*) – Units (the default is "")
- **max** (*float, optional*) – Histogram maximum value (the default is None)
- **min** (*float, optional*) – Histogram minimum value (the default is 0)

`utils.`**GetMaximum**(*hists*)

Get maximum value (i.e. value of 'tallest' bin) of a list of histograms.

Parameters **hists** (*[TH1]*) – List of histograms

Returns Maximum value

Return type float

utils.**GetMinimum**(*hists*)
Get minimum value (i.e. value of 'shortest' bin) of a list of histograms.

Parameters **hists** (*[TH1]*) – List of histograms

Returns Minimum value

Return type float

utils.**GetTChain**(*filenames*, *treename*)
Get TChain (list of Root files containing the same tree)

Parameters

- **filenames** (*[str]*) – Name(s) of ROOT file(s)
- **treename** (*str*) – Name of TTree

Returns The TTree or TChain

Return type TTree or TChain

utils.**GetTTree**(*filename*, *treename*)
Get TTree from file(s).

Returns the TTree if reading a single file or a TChain if reading multiple files from a directory. Exits if file or tree cannot be read.

Parameters

- **filename** (*str*) – Name of ROOT file or the containing directory
- **treename** (*str*) – Name of TTree

Returns The TTree or TChain

Return type TTree or TChain

utils.**MakeHistogram**(*tree*, *plotname*, *nbins*, *xmin*, *xmax*, *selections=''*, *shift=''*, *label=''*)
Make histogram from a TTree.

Parameters

- **tree** (*TTree*) – The tree from which the histogram is made
- **plotname** (*str*) – The plot name; i.e. the name of the branch (or variable) being plotted
- **nbins** (*int*) – Number of bins
- **xmin** (*float*) – Lower edge of first bin
- **xmax** (*float*) – Upper edge of last bin (not included in last bin)
- **selections** (*str, optional*) – Apply selections. See `TTree::Draw()` at https://root.cern.ch/doc/master/classTTree.html for more information
- **shift** (*str, optional*) – Shift histogram by this amount; subtacts this value from the variable being plotted
- **label** (*str, optional*) – The histogram's label; i.e. the entry that will appear in the legend

Returns The histogram

Return type TH1

utils.**MakeLegend**(*hists*, *xmin=0.65*, *ymin=0.65*, *options='LF'*)
Draw the legend.

Legend drawing options are:

- L: draw line associated with hists' lines
- P: draw polymarker associated with hists' marker
- F: draw a box with fill associated with hists' fill
- E: draw vertical error bar if option "L" is also specified

See https://root.cern.ch/doc/master/classTLegend.html for full details.

Parameters

- **hists** (*[TH1]*) – List of histograms
- **xmin** (*float, optional*) – The x position of the bottom left point of the legend (the default is 0.65)
- **ymin** (*float, optional*) – The y position of the bottom left point of the legend (the default is 0.65)
- **options** (*string, optional*) – Pass these options to TLegend::AddEntry(). Default is "LF"

utils.**MakePad**(*name*, *title*, *xlow*, *ylow*, *xup*, *yup*)
Make a pad.

This function replaces the typical TPad constructor because of an unfortunate quirk of PyROOT that forces you to set the ownership of the Pad when creating it, otherwise it gives a segmentation fault.

See https://root.cern.ch/doc/master/classTPad.html.

Parameters

- **name** (*str*) – Pad name
- **title** (*str*) – Pad title
- **xlow** (*float*) – The x position of the bottom left point of the pad
- **ylow** (*float*) – The y position of the bottom left point of the pad
- **xup** (*float*) – The x position of the top right point of the pad
- **yup** (*float*) – The y position of the top right point of the pad

utils.**NormalizeHistograms**(*hists*, *width=False*)
Normalize a list of histograms to unity.

Parameters

- **hists** (*[TH1]*) – List of histograms
- **width** (*bool, optional*) – If true, the bin contents and errors are divided by the bin width (the default is False)

utils.**SetHistogramFill**(*hist*, *color=None*, *style=None*, *alpha=1*)
Set the histogram fill properties.

If `SetHistogramFill()` is called with no colour specified, the fill colour is set to the same as the histogram's line colour.

See https://root.cern.ch/doc/master/classTAttFill.html for more information on fill properties.

Parameters

- **hist** (*TH1*) – The histogram
- **color** (*int, optional*) – Fill colour. See https://root.cern.ch/doc/master/classTColor.html. If you know the hex code, rgb values, etc., use `ROOT.TColor.GetColor()`
- **style** (*int, optional*) – Fill style; this one's complicated so best to see the ROOT documentation
- **alpha** (*float, optional*) – Fill transparency; between 0 and 1 (the default is 1, i.e. opaque)

utils.**SetHistogramLine**(*hist*, *color=1*, *width=1*, *style=1*, *alpha=1*)

Set the histogram line properties.

See https://root.cern.ch/doc/master/classTAttLine.html for more information on line properties.

Parameters

- **hist** (*TH1*) – The histogram
- **color** (*int, optional*) – Line colour (the default is 1, i.e. black). See https://root.cern.ch/doc/master/classTColor.html. If you know the hex code, rgb values, etc., use `ROOT.TColor.GetColor()`
- **width** (*int, optional*) – Line width in pixels; between 1 and 10 (the default is 1)
- **style** (*int, optional*) – Line style; between 1 and 10 (the default is 1, i.e. solid line)
- **alpha** (*float, optional*) – Line transparency; between 0 and 1 (the default is 1, i.e. opaque)

utils.**SetYRange**(*hists*, *max=None*, *min=0*)

Set the y-axis range (max and min) on a list of histograms.

If the max value is not provided, it calls *GetMaximum()* to get the maximum value from the list of histograms

Parameters

- **hists** (*[TH1]*) – List of histograms
- **max** (*float, optional*) – Max value (the default is None)
- **min** (*float, optional*) – Min value (the default is 0)

CHAPTER

FOUR

CONFIG FILE READER

The config files are written in the TOML format.

4.1 Colormaps

I'm a big fan of matplotlib's colourmaps, and I often use the 'tab10' qualitative colourmap when plotting multiple datasets on the same axes. Colours in the config files are specified using hex codes, so I've summarized tab10's hex codes in the table below.

Colour	Hex Code
Blue	#1F77B4
Orange	#FF7F0E
Green	#2CA02C
Red	#D62728
Purple	#9467BD
Brown	#8C564B
Pink	#E377C2
Grey	#7F7F7F
Olive	#BCBD22
Cyan	#17BECF

You can also look up these hex codes and many others directly from matplotlib:

```
>>> import matplotlib
>>> cmap = matplotlib.colors.get_named_colors_mapping()
>>> cmap['tab:blue']
'#1f77b4'
>>> cmap['tab:orange']
'#ff7f0e'
```

and so on.

config_reader.**read**(*config_file*)

Read the plotting configuration parameters into a dictionary.

Parameters **config_file** (*str*) – Path to the config file

Returns Dictionary of configuration parameters

Return type dict

CHAPTER

FIVE

SLIM TREES

A quick and dirty script for reducing the file size of a ROOT file by removing unwanted branches from a TTree.

Examples

The following examples assume the branches you want included in the slimmed tree are listed in `slimbranches.txt`.

Slim the tree `tree` in `data.root` and save to `data.slim.root`:

```
$ slim-trees tree -i data.root
```

Slim the tree `tree` in `data.root` and save to `newdata.root`:

```
$ slim-trees tree -i data.root -o newdata.root
```

Slim the tree `tree` in `data.root`, save to `data.slim.root`, and only keep the first 100 events:

```
$ slim-trees tree -i data.root -t 100
```

Note: A standard ROOT installation provides a program called `rootslimtree` which attempts to do much the same that this script does, but is more feature-rich. The source code is available in the main ROOT git repository.

There are some quirks about `rootslimtree` that I don't understand, for example I've seen it *increase* the file size of ROOT file after stripping away unwanted branches. This is obviously not ideal.

CHAPTER

SIX

FORMATTING CONSOLE OUTPUT

A collection of utilities for formatting console output with colours.

This module is mostly used in the inner workings of **atlasplots** utility functions but you can use it in your plotting scripts too!

Supported Colours

Colour	Command
Red	`bcolor.red`
Green	`bcolor.green`
Yellow	`bcolor.yellow`
Blue	`bcolor.blue`
Magenta	`bcolor.magenta`
Cyan	`bcolor.cyan`
White	`bcolor.white`

Colour aliases

Command	Colour
`bcolor.HEADER`	Blue
`bcolor.OK`	Green
`bcolor.WARNING`	Yellow
`bcolor.ERROR`	Red
`bcolor.FATAL`	Red

Pre-formatted Strings

Command	Returns
`bcolor.ok()`	`OK` (in green)
`bcolor.warning()`	`Warning` (in yellow)
`bcolor.error()`	`Error` (in red)

Other Text Formatting

Style	Command
Bold	`bcolor.bold`
Faint Text	`bcolor.faint`
Italic	`bcolor.italic`
Underline	`bcolor.underline`

Note: Always end the string you want formatted with `bcolor.end`.

Examples

Import the `bcolor` class:

```
>>> from atlasplots.console import bcolor
```

Print text with colour:

```
>>> print(bcolor.blue + "Blue" + bcolor.end)
>>> print(bcolor.red + "Red " + bcolor.green + "Green" + bcolor.end)
```

You can also use colour aliases for printing OK/Warning/Error messages:

```
>>> print(bcolor.OK + "OK" + bcolor.end)
>>> print(bcolor.WARNING + "Warning" + bcolor.end)
>>> print(bcolor.ERROR + "Error" + bcolor.end)
```

Or similarly using Python's string formatting:

```
>>> print("{}  Something went wrong!".format(bcolor.ERROR + "Error" + bcolor.end))
```

Or you can use the pre-formatted strings as a shortcut:

```
>>> print("{}  Something went wrong!".format(bcolor.error()))
```

CHAPTER

SEVEN

EXAMPLES

7.1 Random Histogram

This module plots a random histogram using the ATLAS Style.

```
from __future__ import absolute_import, division, print_function

import ROOT as root

from atlasplots import atlas_style as astyle


def main():
    # Set the ATLAS Style
    astyle.SetAtlasStyle()

    # Construct the canvas
    c1 = root.TCanvas("c1", "The FillRandom example", 0, 0, 800, 600)

    # Define a distribution
    form1 = root.TFormula("form1", "abs(sin(x)/x)")
    sqroot = root.TF1("sqroot", "x*gaus(0) + [3]*form1", 0, 10)
    sqroot.SetParameters(10, 4, 1, 20)

    # Randomly fill the histrogram according to the above distribution
    hist = root.TH1F("hist", "Test random numbers", 100, 0, 10)
    hist.FillRandom("sqroot", 10000)
    hist.Draw()

    # Set axis titles
    hist.GetXaxis().SetTitle("x axis")
    hist.GetYaxis().SetTitle("y axis")

    # Add the ATLAS Label
    astyle.ATLASLabel(0.2, 0.87, "Internal")

    # Save the plot as a PDF
    c1.Update()
    c1.Print("random_hist.pdf")


if __name__ == '__main__':
    main()
```

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX