vzlog Documentation

Release 0.1.8.git

Gustav Larsson

September 04, 2015

Contents

Contents:

CHAPTER 1

Tutorial

The main class of this package is *VzLog*, which manages an HTML output.

1.1 Creating a VzLog object

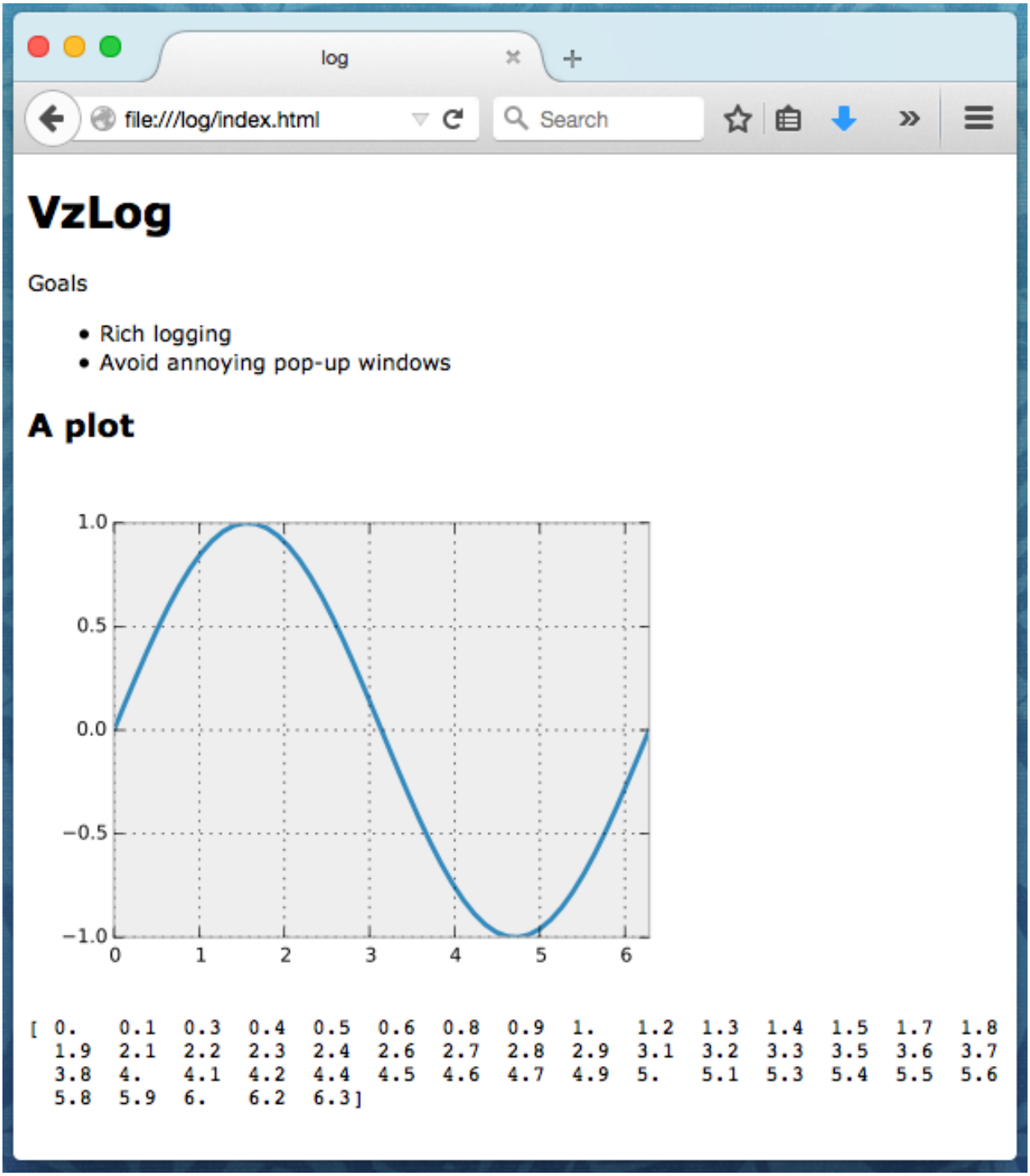
To start a log saving to the folder *log*, construct a new object as follows:

```
import vzlog
vz = vzlog.VzLog('log')
```

Refer to the *VzLog* for outputting functions. Here is an example:

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

vz = vzlog.VzLog('log')

vz.title('VzLog')
vz.text('Goals')
vz.items(['Rich logging', 'Avoid annoying pop-up windows'])

vz.section('A plot')
x = np.linspace(0, 2*np.pi)
fig = plt.figure(figsize=(4, 3))
plt.plot(x, np.sin(x))
plt.xlim((0, 2*np.pi))
plt.savefig(vz.impath('svg'))

np.set_printoptions(precision=1)
vz.log(x)
```

With the output saved to `log/index.html`:



1.2 Using the default `VzLog` object

Instead of manually creating a VzLog instance, it is more common to implement one that sets its path automatically:

```
>>> from vzlog.default import vz
```

This will draw information from the following environment variables:

VZ_DIR Directory of all your VzLog outputs.

VZ_NAME Directory name of your output.

VZ_FILE_RIGHTS File rights of all your images.

The path of a default VzLog object is constructed by joining `VZ_DIR` and `VZ_NAME`. This means that it is easy to keep a folder with many different plotting documents. Here is an example where a new directory will be used:

```
$ VZ_NAME=simple python examples/simple_test.py
```

You can also set these up more permanently by adding them to your `~/.bashrc`:

```
export VZ_DIR=~/html
export VZ_NAME=plot
export VZ_FILE_RIGHTS=0775
```

In this example, your document file will be placed in `~/html/plot/index.html` and the file rights 0775 mean that user/group can read, write and execute and the rest can read and execute.

1.3 Plotting with matplotlib

When you are plotting directly to file, you want to tell matplotlib not to start an interactive session. You can do this as follows:

```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

Since this is such a common use case in vzlog, the following short-hand will do this automatically:

```
import vzlog.pyplot as plt
```

This will also tweak the rendering style.

CHAPTER 2

API Reference

2.1 VzLog

class vzlog.**VzLog**(*path*, *name=None*, *file_rights=None*, *encoding='utf-8'*)

Logging class that manages an HTML log file. Mainly used for visually rich logging with plenty of images.

Parameters

- **path** – Path the directory where the files will be saved.
- **name** – Specify name of the output document. This will be used to set the document title. Inferred from *path* if set to *None*.

clear()

Prepares a folder for logging. This also clears any previous output and can thus be used during an interactive session when wanting a clean slate.

flush()

Flushes the output. This mainly updates the file rights for newly added files. Normally, you do not need to call this yourself. However, it can be useful during an interactive session when the file rights have not be processed yet.

impath(*ext='png'*, *scale=1.0*)

This generates a path to an image file, outputs the image and returns the path. Specify the extension with *ext*.

```
>>> from vzlog.default import vz
```

We can output an image by saving the file:

```
>>> ag.image.save(vz.impath(ext='png'), im)
```

Or with matplotlib using a vector format:

```
>>> import pylab as pl
>>> pl.figure()
>>> pl.plot([1,3,2,3,1])
>>> pl.savefig(vz.impath(ext='svg'))
```

Parameters

- **ext** – Extension of image file.
- **scale** – Scale of image when viewed in the browser. This will use nearest neighbor upscaling that works well with pixel grids (if your browser supports it).

items (*items*, *style='bullets'*)
Outputs an item list.

Parameters

- **items** – Iterable of representable objects. If an item is a list, it will call itself recursively.
- **style** – List style: `'bullets'` or `'numbers'`. Use a list/tuple of values if you want different levels to have different styles.

log (**args*, ***kwargs*)
Logs text with mono-spaced font (`<pre>`). This function and many other text outputting functions uses the same arguments as the built-in *print* function. The only difference is that will ignore `file` if specified.

```
>>> vz.log('Logging a value', 100)
```

output (*obj*)
Outputs an object. This will look for *obj._vzlog_output_* and pass itself along as the only argument. If the object does not have such a function, it will fall back to calling *VzLog.log*.

output_with_tag (*tag*, **args*, ***kwargs*)
Outputs strings surrounded by a specific HTML tag.

savefig (*fig=None*)
Saves a matplotlib figure to the log file. This can also be done using *impath*, but this will save both a SVG and a PDF version. The SVG will be viewed directly in the browser next to a link to the PDF (so it can be downloaded).

Parameters **fig** – If specified, this should a matplotlib figure object that `fig.savefig` will be called on.

section (**args*, ***kwargs*)
Adds a section title (`<h2>`). See *log* for arguments.

text (**args*, ***kwargs*)
Adds a paragraph of text (`<p>`). See *log* for arguments.

timed (**args*, ***kwds*)
Context manager to make it easy to time the execution of a piece of code. This timer will never run your code several times and is meant for simple in-production timing, instead of benchmarking. It measure wall-clock time.

```
>>> with vz.timed('Sleep'):
...     time.sleep(1)
```

Parameters **name** – Name of the timing block, to identify it.

title (**args*, ***kwargs*)
Adds a title (`<h1>`). See *log* for arguments.

2.2 Images (image)

Functions for handling and generating images.

class vzlog.image.**ImageGrid**(*data=None*, *rows=None*, *cols=None*, *shape=None*, *border_color=1*, *border_width=1*, *cmap=None*, *vmin=None*, *vmax=None*, *vsym=False*, *global_bounds=True*)
An image grid used for combining equally-sized intensity images into a single larger image.

Parameters

- **data** (*ndarray, ndim in [2, 3, 4]*) – The last two axes should be spatial dimensions of an intensity patch. The rest are used to index them. If *ndim* is 2, then a single image is shown. If *ndim* is 3, then *rows* and *cols* will determine its layout. If *ndim* is 4, then *rows* and *cols* will be ignored and the grid will be layed out according to its first two axes instead. If data is set to None, then an empty image grid will be initialized. In this case, rows and cols are both required.
- **rows/cols** (*int or None*) – The number of rows and columns for the grid. If both are None, the minimal square grid that holds all images will be used. If one is specified, the other will adapt to hold all images. If both are specified, then it possible that the grid will be vacuous or that some images will be omitted.
- **shape** (*tuple*) – Shape of the each grid image. Only use if `data` is set to *None*, since it should otherwise be inferred from the data.
- **border_color** (*float or np.ndarray of length 3*) – Specify the border color as an array of length 3 (RGB). If a scalar is given, it will be interpreted as the grayscale value.
- **border_width** – Border with in pixels. If you rescale the image, the border will be rescaled with it.
- **cmap/vmin/vmax/vsym** – See *ImageGrid.set_image*.
- **global_bounds** (*bool*) – If this is set to True and either *vmin* or *vmax* is not specified, it will infer it globally for the data. If *vsym* is True, the global bounds will be symmetric around zero. If it is set to False, it determines range per image, which would be the equivalent of calling *set_image* manually with *vmin*, *vmax* and *vsym* set the same.

Examples

```
>>> import vzlog
>>> import numpy as np
>>> import matplotlib.pylab as plt
>>> from matplotlib.pylab import cm
>>> rs = np.random.RandomState(0)
```

Let's generate a set of 100 8x8 image patches.

```
>>> shape = (100, 8, 8)
>>> data = np.arange(np.prod(shape)).reshape(shape)
>>> data += rs.uniform(0, np.prod(shape), size=shape)
```

Creating the image grid:

```
>>> grid = vzlog.image.ImageGrid(data, cmap=cm.hsv)
>>> img = grid.scaled_image(scale=5)
>>> plt.imshow(img)
>>> plt.show()
```

If you are working in an IPython notebook, you can display `img` simply by adding it to the end of a cell.

image
Returns the image as a skimage.io.Image class.

save (*path*, *scale=1*)
Save the image to file.

Parameters

- **path** (*str*) – Output path.
- **scale** (*int*) – Upscaling using nearest neighbor, e.g. a scale of 5 will make each pixel a 5x5 rectangle in the output.

scaled_image(*scale=1*)

Returns a nearest-neighbor upscaled scaled version of the image.

Parameters **scale** (*int*) – Upscaling using nearest neighbor, e.g. a scale of 5 will make each pixel a 5x5 rectangle in the output.

Returns **scaled_image** – Returns a scaled up RGB image. If you do not have scikit-image, it will be returned as a regular Numpy array. The benefit of wrapping it in *Image*, is so that it will be automatically displayed in IPython notebook, without having to issue any drawing calls.

Return type skimage.io.Image, (height, width, 3)

set_image(*image*, *row*, *col*, *cmap=None*, *vmin=None*, *vmax=None*, *vsym=False*)

Sets the data for a single window.

Parameters

- **image** (*ndarray, ndim=2*) – The shape should be the same as the *shape* specified when constructing the image grid.
- **row/col** (*int*) – The zero-index of the row and column to set.
- **cmap** (*cmap (from matplotlib.pylab.cm)*) – The color palette to use. Default is grayscale.
- **vmin/vmax** (*numerical or None*) – Defines the range of the color palette. None, which is default, takes the range of the data.
- **vsym** (*bool*) – If True, this means that the color palette will always be centered around 0. Even if you have specified both *vmin* and *vmax*, this will override that and extend the shorter one. Good practice is to specify neither *vmin* or *vmax* or only *vmax* together with this option.

class vzlog.image.**ColorImageGrid**(*data=None*, *rows=None*, *cols=None*, *shape=None*, *border_color=1*, *border_width=1*, *vmin=0.0*, *vmax=1.0*, *vsym=False*, *global_bounds=True*)

An image grid used for combining equally-sized RGB images into a single larger image. It is similar to *ImageGrid*, except it only works for RGB images with color channels scaled in [0, 1].

Parameters

- **data** (*ndarray, ndim in [3, 4, 5]*) – The last three axes should be spatial dimensions and a color channel. The rest are used to index them. If *ndim* is 3, then a single image is shown. If *ndim* is 4, then *rows* and *cols* will determine its layout. If *ndim* is 5, then *rows* and *cols* will be ignored and the grid will be layed out according to its first two axes instead. If data is set to None, then an empty image grid will be initialized. In this case, rows and cols are both required.
- **rows/cols** (*int or None*) – The number of rows and columns for the grid. If both are None, the minimal square grid that holds all images will be used. If one is specified, the other will adapt to hold all images. If both are specified, then it possible that the grid will be vacuous or that some images will be omitted.
- **shape** (*tuple*) – Shape of the each grid image. Only use if `data` is set to *None*, since it should otherwise be inferred from the data.
- **border_color** (*float or np.ndarray of length 3*) – Specify the border color as an array of length 3 (RGB). If a scalar is given, it will be interpreted as the grayscale value.

- **border_width** – Border with in pixels. If you rescale the image, the border will be rescaled with it.
- **cmap/vmin/vmax/vsym** – See *ImageGrid.set_image*.
- **global_bounds** (*bool*) – If this is set to True and either *vmin* or *vmax* is not specified, it will infer it globally for the data. If *vsym* is True, the global bounds will be symmetric around zero. If it is set to False, it determines range per image, which would be the equivalent of calling *set_image* manually with *vmin*, *vmax* and *vsym* set the same.

image
Returns the image as a skimage.io.Image class.

save (*path*, *scale=1*)
Save the image to file.

Parameters

- **path** (*str*) – Output path.
- **scale** (*int*) – Upscaling using nearest neighbor, e.g. a scale of 5 will make each pixel a 5x5 rectangle in the output.

scaled_image (*scale=1*)
Returns a nearest-neighbor upscaled scaled version of the image.

Parameters **scale** (*int*) – Upscaling using nearest neighbor, e.g. a scale of 5 will make each pixel a 5x5 rectangle in the output.

Returns **scaled_image** – Returns a scaled up RGB image. If you do not have scikit-image, it will be returned as a regular Numpy array. The benefit of wrapping it in *Image*, is so that it will be automatically displayed in IPython notebook, without having to issue any drawing calls.

Return type skimage.io.Image, (height, width, 3)

set_image (*image*, *row*, *col*, *vmin=0.0*, *vmax=1.0*, *vsym=False*)
Sets the data for a single window.

Parameters

- **image** (*ndarray, ndim=3*) – The shape should be the same as the *shape* specified when constructing the image grid, plus an axis of length 3 for the color channels.
- **row/col** (*int*) – The zero-index of the row and column to set.
- **vmin/vmax** (*numerical or None*) – Defines the range of the color palette. None, takes the range of the data. Default is vmin=0 and vmax=1, for plotting normal RGB images.
- **vsym** (*bool*) – If True, this means that the color palette will always be centered around 0. Even if you have specified both *vmin* and *vmax*, this will override that and extend the shorter one. Good practice is to specify neither *vmin* or *vmax* or only *vmax* together with this option.

CHAPTER 3

Indices and tables

- genindex
- modindex

V

C

F

I

L

O

S

T

V