ipyscales Documentation

Release 0.3.1.dev

Vidar Tonaas Fauske

Nov 08, 2018

Installation and usage

Version: 0.3.1.dev

A Jupyter widgets library for scales.

CHAPTER 1

Quickstart

To get started with ipyscales, install with pip:

```
pip install ipyscales
```

or with conda:

```
conda install ipyscales
```

CHAPTER 2

Contents

2.1 Installation

The simplest way to install ipyscales is via pip:

```
pip install ipyscales
```

or via conda:

```
conda install ipyscales
```

If you installed via pip, and notebook version < 5.3, you will also have to install / configure the front-end extension as well. If you are using classic notebook (as opposed to Jupyterlab), run:

```
jupyter nbextension install [--sys-prefix / --user / --system] --py ipyscales

jupyter nbextension enable [--sys-prefix / --user / --system] --py ipyscales
```

with the appropriate flag. If you are using Jupyterlab, install the extension with:

```
jupyter labextension install @jupyter-widgets/jupyterlab-manager --no-build

jupyter labextension install jupyter-colorbar
```

If you are installing using conda, these commands should be unnecessary, but If you need to run them the commands should be the same (just make sure you choose the *–sys-prefix* flag).

2.2 Examples

This section contains several examples generated from Jupyter notebooks. The widgets have been embedded into the page for demonstrative purposes.

2.2.1 Introduction

```
In [1]: import ipyscales
```

```
In [9]: # Make a default scale, and list its trait values:
        scale = ipyscales.LinearScale()
        print(', '.join('%s: %s' % (key, getattr(scale, key)) for key in sorted(scale.keys) if not ke
```

```
clamp: False, domain: (0.0, 1.0), interpolator: interpolate, range: (0.0, 1.0)
```

2.2.2 Color maps

```
In [1]: from ipyscales import (
            LinearColorScale, LogColorScale, NamedSequentialColorMap, NamedDivergingColorMap,
            NamedOrdinalColorMap, ColorBar
        )
```

```
In [2]: from ipyscales._example_helper import use_example_model_ids
        use_example_model_ids()
```

```
In [3]: from IPython.display import display

        def visualize(cm):
            display(ColorBar(
                colormap=cm,
                length=500,
                orientation='horizontal'
            ))
```

```
In [4]: visualize(
            LinearColorScale(range=('red', 'blue'))
        )
```

```
ColorBar(colormap=LinearColorScale(domain=(0.0, 1.0), range=('red', 'blue')), length=500, orientatior
```

```
In [5]: visualize(
            LogColorScale(range=('green', 'orange'))
        )
```

```
ColorBar(colormap=LogColorScale(domain=(1.0, 10.0), range=('green', 'orange')), length=500, orientati
```

```
In [6]: visualize(
            NamedSequentialColorMap('Viridis')
        )
```

```
ColorBar(colormap=NamedSequentialColorMap(domain=(0.0, 1.0)), length=500, orientation='horizontal')
```

```
In [7]: visualize(
            NamedDivergingColorMap('RdBu', domain=(-1, 0, 1))
        )
```

```
ColorBar(colormap=NamedDivergingColorMap(domain=(-1.0, 0.0, 1.0), name='RdBu'), length=500, orientati
```

```
In [8]: visualize(
            NamedOrdinalColorMap('Accent', domain=tuple(range(10)))
        )
```

```
ColorBar(colormap=NamedOrdinalColorMap(cardinality=8, domain=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9), name='Ac
```

Example use with image

Generate some 2D data to use as image

```
In [9]: import numpy as np

        delta = 0.025
        x = y = np.arange(-3.0, 3.0, delta)
        X, Y = np.meshgrid(x, y)
        Z1 = np.exp(-X**2 - Y**2)
        Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
        Z = (Z1 - Z2) * 2
```

Set up the scale to use, the color transform, and the image:

```
In [10]: from ipyscales.datawidgets import ScaledArray
         from ipydatawidgets import DataImage

         scale = NamedSequentialColorMap(domain=(np.min(Z), np.max(Z)))
         data = ScaledArray(data=Z, scale=scale, output_dtype='uint8')
         img = DataImage(data=data)
```

Finally, display the image side-by-side with a color bar, and add a color map selector box underneath:

```
In [11]: from ipywidgets import HBox, VBox

         VBox([
             HBox([img, ColorBar(colormap=data.scale, length=len(x))]),
             scale.edit()
         ])
```

```
VBox(children=(HBox(children=(DataImage(data=ScaledArray(data=array([[  3.04599342e-08,   3.53672816e
```

```
In [ ]:
```

2.2.3 Color bars

ipyscales includes widgets for visualising color maps as color bars, and for manually editing color maps.

Let's define two simple color maps first:

```
In [1]: import ipyscales
```

```
In [2]: cm = ipyscales.LinearColorScale(
            range=('blue', 'green'),
            domain=(1000, 5000)
        )
        cm_div = ipyscales.LinearColorScale(
            range=('blue', 'white', 'red'),
            domain=(-1, 0, 1)
        )
```

Next, let us define a color bar for the first color map:

```
In [3]: colorbar = ipyscales.ColorBar(colormap=cm, length=300)
        assert colorbar.colormap is cm
```

```
In [4]: colorbar
```

```
ColorBar(colormap=LinearColorScale(domain=(1000.0, 5000.0), range=('blue', 'green')), length=300)
```

The orientation of the color bar can be made horizontal as well:

```
In [5]: ipyscales.ColorBar(colormap=cm_div, length=300, orientation='horizontal')
```

```
ColorBar(colormap=LinearColorScale(domain=(-1.0, 0.0, 1.0), range=('blue', 'white', 'red')), length=
```

The color map can include transparent colors, and the color bar also supports this:

```
In [6]: cm_div_transp = ipyscales.LinearColorScale(
            range=('rgba(0, 0, 255, 0.5)', 'white', 'rgba(255, 0, 0, 0.5)'),
            domain=(-1, 0, 1)
        )
```

```
In [7]: ipyscales.ColorBar(colormap=cm_div_transp, length=200)
```

```
ColorBar(colormap=LinearColorScale(domain=(-1.0, 0.0, 1.0), range=('rgba(0, 0, 255, 0.5)', 'white',
```

The editor `ColorMapEditor` can be used to manually edit the color map. The actions you can do are: - Change the color of a stop by double-clicking the corresponding handle. - Change the location of stops by dragging the handles.

```
In [8]: ipyscales.ColorMapEditor(
            colormap=cm_div_transp, length=400, orientation='horizontal')
```

```
ColorMapEditor(colormap=LinearColorScale(domain=(-1.0, 0.0, 1.0), range=('rgba(0, 0, 255, 0.5)', 'whi
```

Future features planned for the editor include: - Adding new stops. - Removing stops. - Modifying the stop location by numeric input. - UI for changing opacity of a color stop.

```
In [ ]:
```

2.2.4 Scaled data widget

ipyscales also allow for front-end based scaling of array data by implementing the ipydatawidgets interfaces. This allows e.g. for sending you (large) dataset once, and then rapidly re-scaling it on the front-end side.

```
In [1]: import numpy as np
        from ipyscales import LinearScale, LogScale
        from ipyscales.datawidgets import ScaledArray
        from ipydatawidgets import DataImage
        np.random.seed(0)
```

Set up some random RGBA data:

```
In [2]: data = np.array(255 * np.random.rand(200, 200, 4), dtype='uint8')
```

Set up two simple scales for translating each channel value, one linear, one logarithmic:

```
In [3]: linear = LinearScale(range=(0, 255), domain=(0, 255), clamp=True)
        # Setup log scale as alternative scale. Note the non-zero domain with clamp!
        log = LogScale(range=(0, 255), domain=(1, 255), clamp=True)
```

Set up our scaled data source. This will not synchronize the scaled data, but if passed to a data union trait, the scaled data will be used:

```
In [4]: scaled_data = ScaledArray(data=data, scale=linear)
```

Pass the scaled data to a `DataImage` to visualize it:

```
In [5]: image = DataImage(data=scaled_data)
        image
```

```
DataImage(data=ScaledArray(data=array([[[139, 182, 153, 138],
        [108, 164, 111, 227],
        [245,  97,...
```

Add some controls for selecting which scale to use, and what the range of the scales should be:

```
In [6]: from ipywidgets import FloatRangeSlider, jslink, HBox
        from ipyscales.selectors import WidgetDropdown

        range_selector = FloatRangeSlider(min=0, max=255, step=1, description='range', readout_format
        jslink((linear, 'range'), (range_selector, 'value'))
        jslink((range_selector, 'value'), (log, 'range'))
```

```
        scale_selector = WidgetDropdown(options={'Linear': linear, 'Log': log}, description='Scale')
        jslink((scale_selector, 'value'), (scaled_data, 'scale'))

        HBox([scale_selector, range_selector])
```

```
HBox(children=(WidgetDropdown(options={'Linear': LinearScale(clamp=True, domain=(0.0, 255.0), range=
```

Color mapped data

Set up a 2D scalar field of floats (0-1) to act as our data. Here, a nice diagonal gradient:

```
In [7]: side_length = 200
        scalar_field = np.sum(np.meshgrid(
            np.linspace(0, 0.5, side_length),
            np.linspace(0, 0.5, side_length),
        ), axis=0)
```

This time, we use a color map as the scale. Here we use named, sequential color maps:

```
In [8]: from ipyscales import NamedSequentialColorMap
        cmap = NamedSequentialColorMap(name='viridis')
```

When setting up a scaled array with a color map, we should specify the output dtype we want. If not, it will inherit the dtype of the input data. In this case, the input dtype is float, and `DataImage` expects a unsigned 8-bit integer.

```
In [9]: mapped_data = ScaledArray(data=scalar_field, scale=cmap, output_dtype='uint8')
```

Visualize the mapped data:

```
In [10]: mapped_image = DataImage(data=mapped_data)
         mapped_image
```

```
DataImage(data=ScaledArray(data=array([[ 0.        ,  0.00251256,  0.00502513, ...,  0.49497487,
         0.49...
```

Show a drop-down selector for the color map name:

```
In [11]: cmap.edit()
```

```
VBox(children=(StringDropdown(options=('Viridis', 'Inferno', 'Magma', 'Plasma', 'Warm', 'Cool', 'Cube
```

```
In [ ]:
```

2.3 Developer install

To install a developer version of ipyscales, you will first need to clone the repository:

```
git clone https://github.com/vidartf/ipyscales
cd ipyscales
```

Next, install it with a develop install using pip:

```
pip install -e .
```

If you are planning on working on the JS/frontend code, you should also do a link installation of the extension:

```
jupyter nbextension install [--sys-prefix / --user / --system] --symlink --py␣
↪ipyscales

jupyter nbextension enable [--sys-prefix / --user / --system] --py ipyscales
```

with the appropriate flag. Or, if you are using Jupyterlab:

```
jupyter labextension install @jupyter-widgets/jupyterlab-manager --no-build
jupyter labextension install ./js
```