Shendruk Group Plotting Styles and Colours

Tyler Shendruk

June 21, 2021

***This is a non-official guide and I expect that over time the group will add to and improve on what is currently here.

I have created an `mplstyle` sheet titled `shendrukGroupStyle.mplstyle` for matplotlib. This style file just needs to be placed in the matplotlib libraries and then loaded at the start of all python plotting scripts. This style sheet sets the default but you still have all the control over matplotlib and can reset colours, sizes, linewidths, *etc.* as you always do in matplotlib. As an example, I've included a few simple plots below.

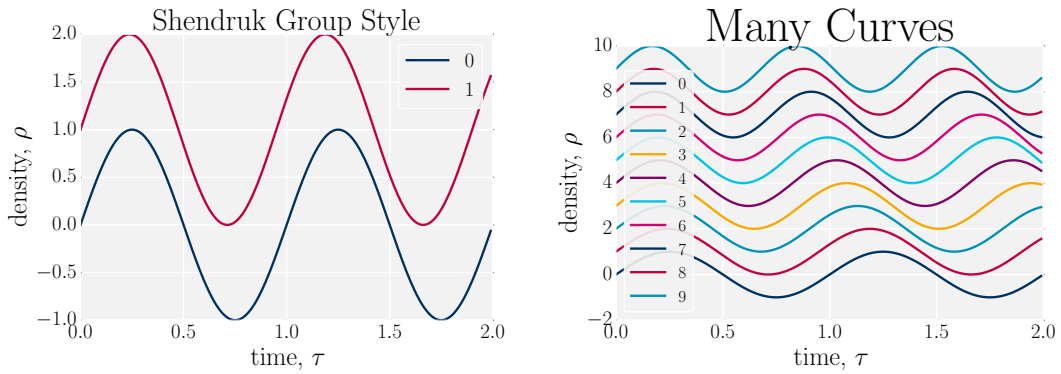


Figure 1: Default plotting on left. Modified title and legend size on right.

The colours correspond to the colours Edinburgh brand colours. I have included seven new 'default' colours that will be automatically cycled through by matplotlib. This is shown in the second panel with many curves. As before, one can override the default and change any of the options. For instance here, I made a legend with a frame and a white background (rather than the default of no frame and transparent background) because there are so many curves. To further illustrate, I increased the title-font size too. Colours, font size, everything can still be changed in the normal ways.

Cyclic Colours	
Name	R, G, B
saphire	**0, 50, 95**
crimson	**193, 0, 67**
capri	**0, 196, 223**
amber	**244, 170, 0**
plum	**129, 2, 98**
cerulean	**0, 145, 181**
ruby	**212, 0, 114**

Table 1: The seven cyclic colours from the official Edinburgh brand included in the `shendrukGroupStyle.mplstyle` and `shendrukGroupFormat.py` files.

Edinburgh's brand include additional colours than those included in the cyclic colours. These can be found in the table below.

Other Official Colours	
cardinal	**172, 0, 64**
cinnamon	**205, 90, 19**
limegreen	**41, 188, 41**
gold	**141, 116, 74**
taupe	**110, 80, 72**
teal	**69, 126, 129**
forestgreen	**0, 70, 49**
mahogany	**106, 51, 40**
silver	**194, 211, 223**
oldrose	**184, 133, 141**
curry	**156, 154, 0**
cobalt	**0, 80, 114**

Table 2: The non-cyclic colours from the official Edinburgh brand included in `shendrukGroupFormat.py` files.

Finally, I also made a few modified colours (which I found useful in creating colourmaps; see below).

Modified Colours	
rubydarker	**197, 0, 99**
purple	**56, 6, 92**
cardinaldarker	**97, 0, 36**
ceruleandarker	**0, 113, 140**
amberlighter	**240, 191, 79**
amberbrighter	**245, 242, 88**
white	255, 255, 255
onyx	**15, 15, 15**
bggrey	**245, 245, 245**

Table 3: The modified colours in `shendrukGroupFormat.py` files.

To make these conveniently accessible, I've put these into a python file called `shendrukGroupFormat.py` that can be imported. Not only can we access any of the colours (to use in plots *etc.* but there are convenient `classes` and `functions`. All codes import set the group style and import the group format file:

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('shendrukGroupStyle')
import shendrukGroupFormat as ed
```

As an example of how to use the imported `shendrukGroupFormat.py` file, the following code imports the it as `ed` and then prints the RGB for saphire then uses the list of colours (`clist()`) to print all the cyclic colours and finally prints the RGB for saphire again from the list of cyclic colours.

```
print ed.saphire
print ed.clist().cyclic
print ed.clist().cyclic["saphire"]
```

The output is

```
[0, 0.19607843137254902, 0.37254901960784315]
{'amber': [0.9568627450980393, 0.6666666666666666, 0],
```

```
 'plum': [0.5058823529411764, 0.00784313725490196, 0.3843137254901961],
 'capri': [0, 0.7686274509803922, 0.8745098039215686],
 'cerulean': [0, 0.5686274509803921, 0.7098039215686275],
 'crimson': [0.7568627450980392, 0, 0.2627450980392157],
 'saphire': [0, 0.19607843137254902, 0.37254901960784315],
 'ruby': [0.8313725490196079, 0, 0.4470588235294118]}
 [0, 0.19607843137254902, 0.37254901960784315]
```

This can be done for any of the lists *i.e.*

```
print ed.clist().cyclic
print ed.clist().noncyclic
print ed.clist().modified
print ed.clist().official
print ed.clist().all
```

If you forget what these colours look like, `shendrukGroupFormat.py` contains a convenient table maker, called `plot_colortable()`. The following code will make five colourtables.

```
ed.plot_colortable(ed.clist().cyclic, "Cyclic")
ed.plot_colortable(ed.clist().noncyclic, "Noncyclic")
ed.plot_colortable(ed.clist().modified, "Modified")
ed.plot_colortable(ed.clist().official, "Official")
ed.plot_colortable(ed.clist().all, "All")
plt.show()
```

Figure 2: Colour tables produced by `plot_colortable()` in `shendrukGroupFormat.py`.

If one wants to plot using colours or groups of colours from `shendrukGroupFormat.py` instead of just the automantic cyclic colours, this should now be convenient.

I'll quick put up a few examples of this style will look for other types of graphs.

```
N = 10
for i in ed.clist().official:
    x = np.random.rand(N)
    y = np.random.rand(N)
    colors = np.random.rand(N)
    area = np.pi * (15 * np.random.rand(N))**2
    plt.scatter(x, y, s=area, c=ed.clist().official[i], alpha=0.75, label=i)
```

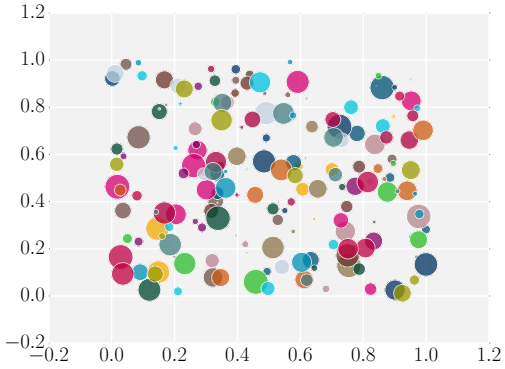


Figure 3: Scatter plot. Notice that scatter plots **do not** cycle so the user must specify colours.

```
x = np.arange(0.1, 4, 0.5)
y = np.exp(-x)
y2 = 1.0-np.exp(-x)
y2err=0.2*y
fig, ax = plt.subplots()
ax.errorbar(x, y, xerr=0.2, yerr=0.4, marker='o')
ax.errorbar(x, y2, yerr=y2err)
ax.errorbar(x, y2-y, yerr=0.4, marker='s')
fig, ax = plt.subplots()
ed.errorbar_fill(x,y,yerr=0.4,marker='o')
ed.errorbar_fill(x,y2,yerr=y2err)
ed.errorbar_fill(x,y2-y,yerr=0.4,marker='s')
```

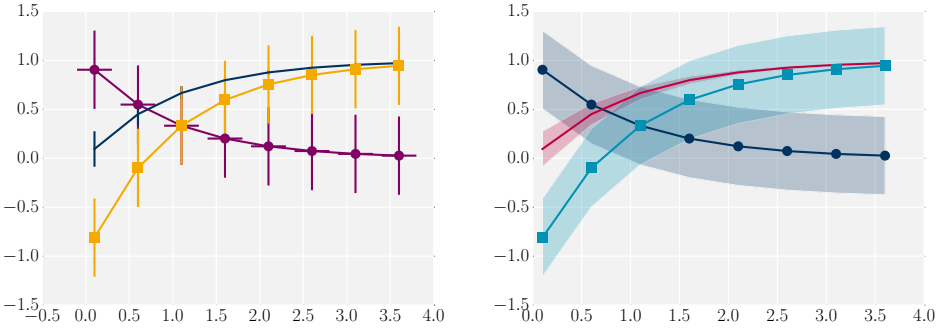


Figure 4: On the left is the default errorbars using the `shendrukGroupFormat` and `shendrukGroupStyle`. On the right, showing the error as a fill between points, which I find far more elegent when we have lots and lots of data points as frequently occurs (I like this better even with fewer).

```
np.random.seed(0)
mu = 10
sigma = 15
x = mu + sigma * np.random.randn(600)
fig, ax = plt.subplots()
n, bins, patches = ax.hist(x, 50, normed=1)
y = mlab.normpdf(bins, mu, sigma)
ax.plot(bins, y, '--o')
ax.set_ylabel('Probability density')
ax.set_title(r'$\mu=100$, $\sigma=15$')
```

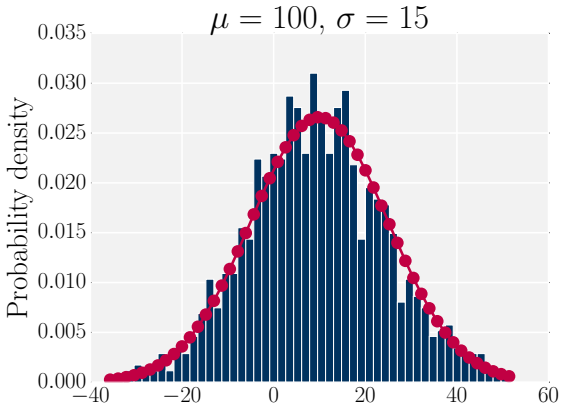


Figure 5: Histogram and fit.

```
dt = 0.001
t = np.arange(0.0, 10.0, dt)
r = np.exp(-t[:1000]/0.05)               # impulse response
x = np.random.randn(len(t))
s = np.convolve(x, r)[:len(x)]*dt  # colored noise

# the main axes is subplot(111) by default
plt.plot(t, s)
plt.axis([0, 1, 1.1*np.amin(s), 2*np.amax(s)])
plt.xlabel('time (s)')
plt.ylabel('current (nA)')
plt.title('Gaussian colored noise')

# this is an inset axes over the main axes
inset1 = plt.axes([.7, .6, .2, .2])
inset1.patch.set_facecolor(ed.silver)
n, bins, patches = plt.hist(s, 400, normed=1,edgecolor=ed.saphire)
plt.title('Probability',fontsize=ed.fontsize)
plt.xticks([])
plt.yticks([])

# this is another inset axes over the main axes
inset2 = plt.axes([0.25, 0.6, .2, .2])
inset2.patch.set_facecolor(ed.silver)
plt.plot(t[:len(r)], r)
plt.title('Impulse',fontsize=ed.fontsize)
plt.xlim(0, 0.2)
```

```
inset2.tick_params(axis="x",labelsize=ed.fontsizeTiny)
inset2.tick_params(axis="y",labelsize=ed.fontsizeTiny)
```

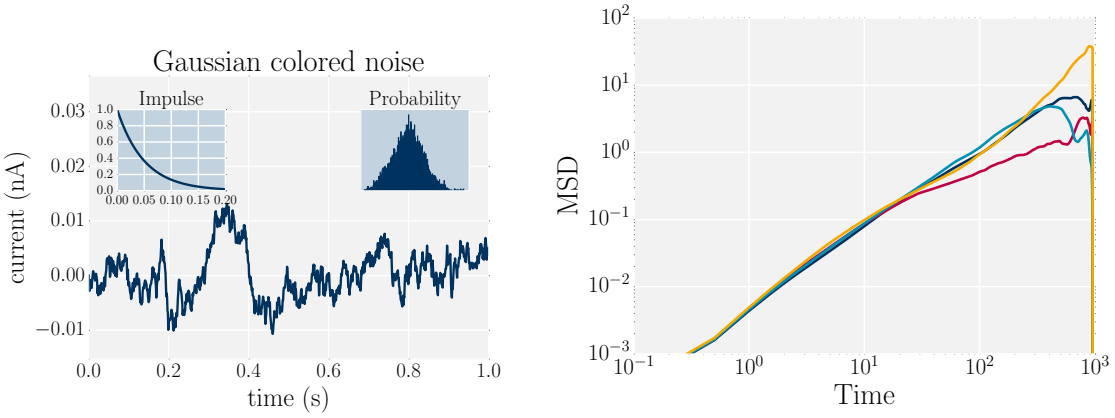
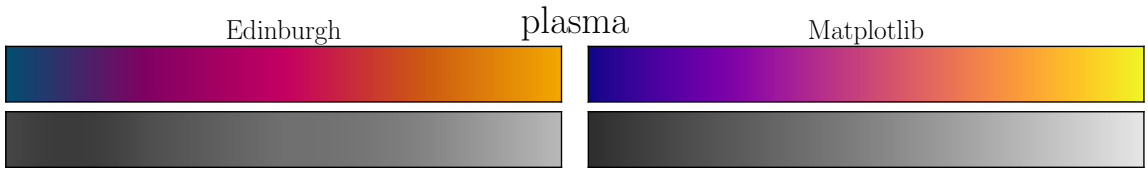


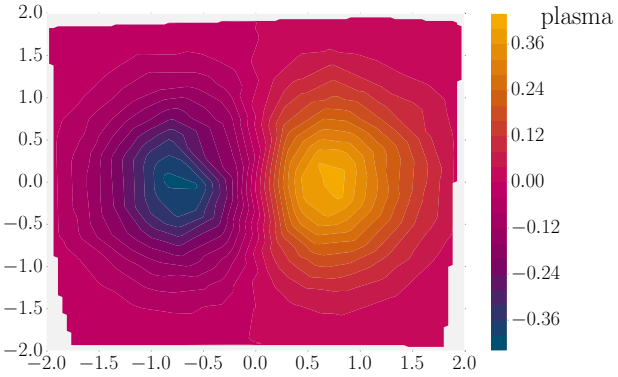
Figure 6: Insets on left. A simple log-log graph on right.

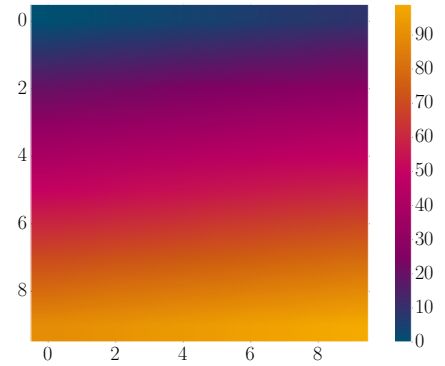
In addition to setting up these individual colours, I've also created a set of colour maps. These colour maps are modelled after standard `matplotlib` colour maps **but** I've used the Edinburgh colours (or slight modifications) to build unique versions of the colour maps for the group. Hopefully, this will give our work a unified but distinct look.

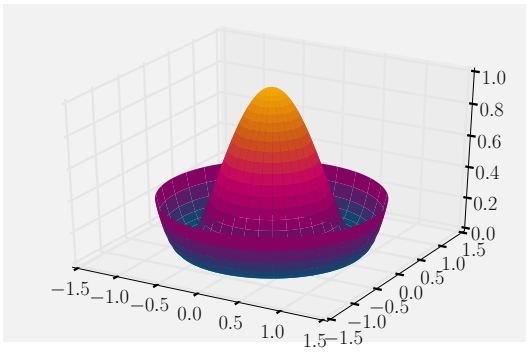
Choosing a proper colour map depends a lot on what you're doing with it — not all of these colour maps are equally good for all purposes. Importantly, many colour maps can put misleading emphasis on different regions because different colours are perceived more than others. The previously ubiquitous colour map `jet` was infamous for this and both `matplotlib` and Matlab have removed it from being the default and replaced it with "perceptually uniform sequential" maps, such as 'viridis', 'plasma', 'inferno' and 'magma'. However, at other times you don't want "perceptually uniform sequential" but rather would like "diverging" which goes from one dark colour through a light colour to another dark colour. We commonly use these for fields like vorticity. See `https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html` for more information with respect to `matplotlib` colour maps.

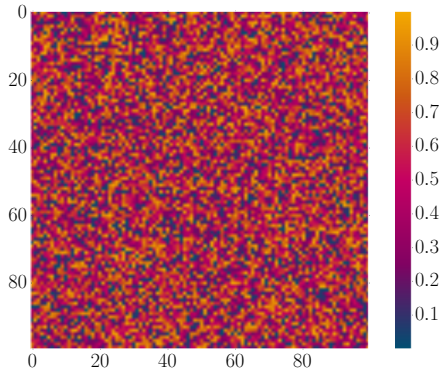
But I've created a set of eight for us to chose from. In what follows, I compare my version to the "standard" matplotlib version then plot six common-ish types of graphs that might use a colour map.

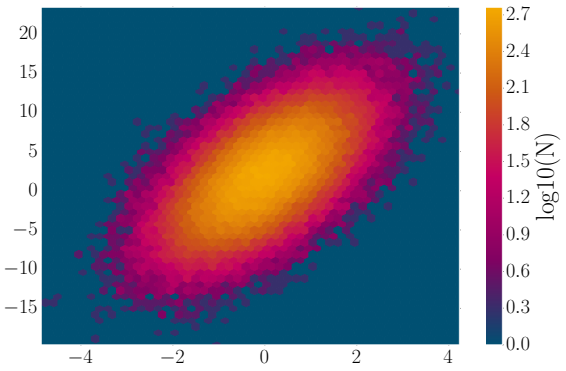

Edinburgh
plasma
Matplotlib

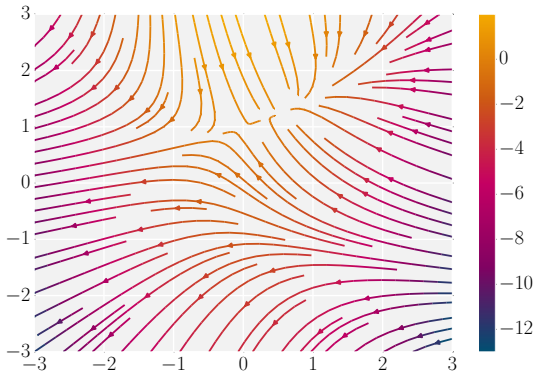

plasma

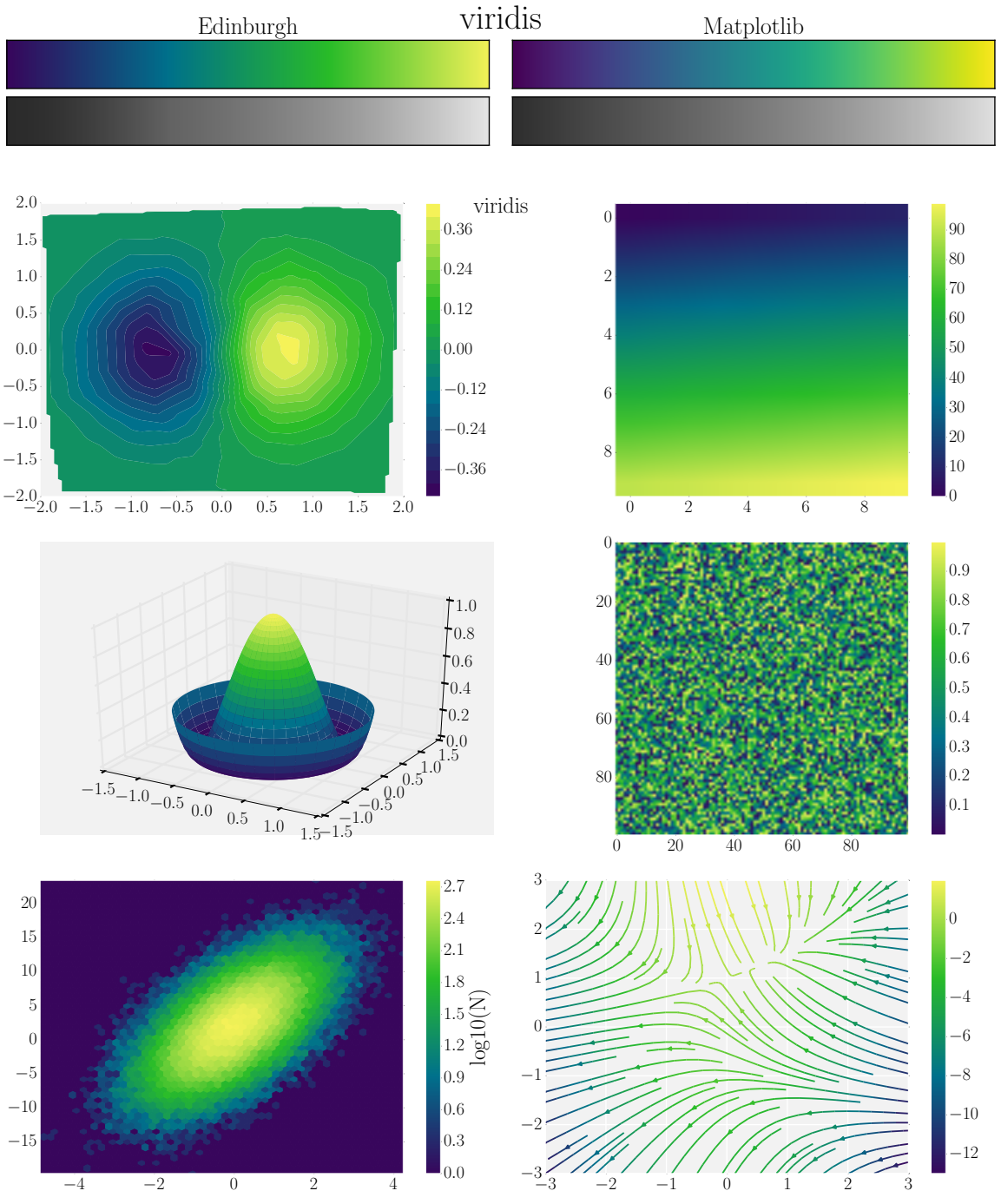


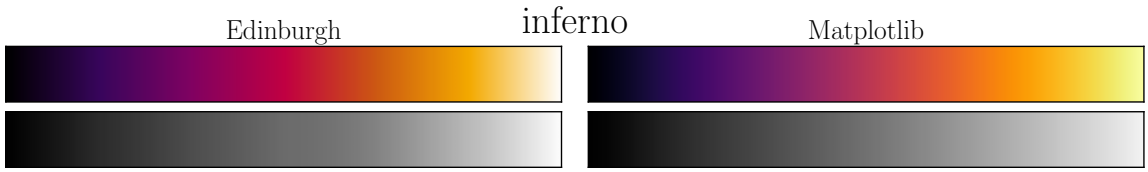


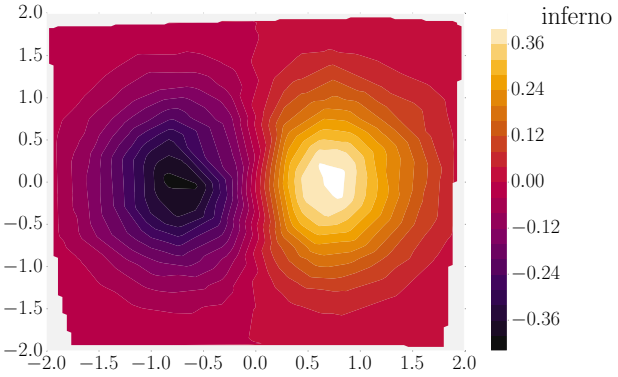


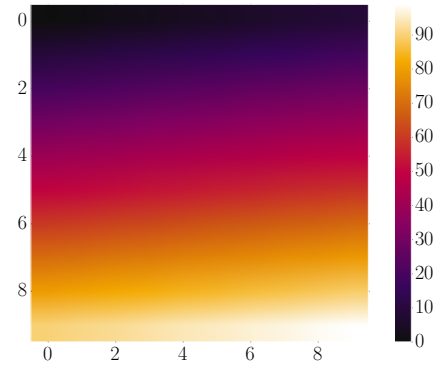

log10(N)

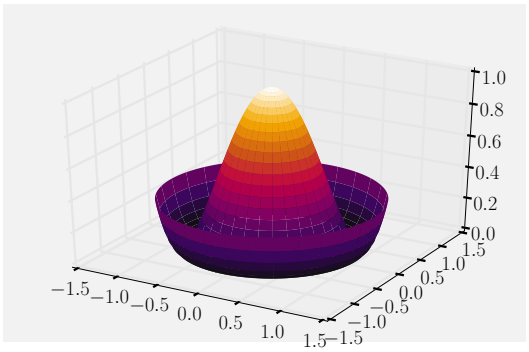


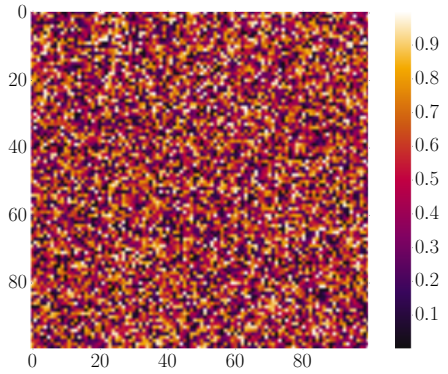

viridis
Edinburgh
Matplotlib
viridis
log10(N)

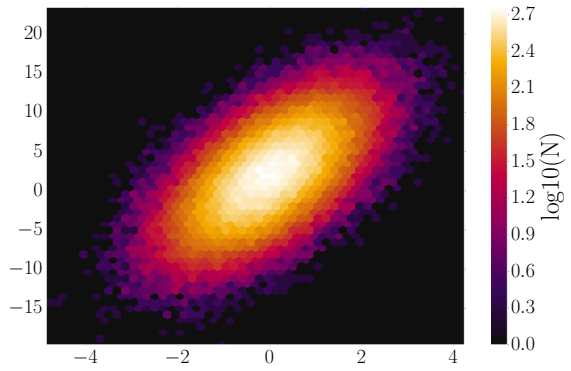

inferno
Edinburgh
Matplotlib

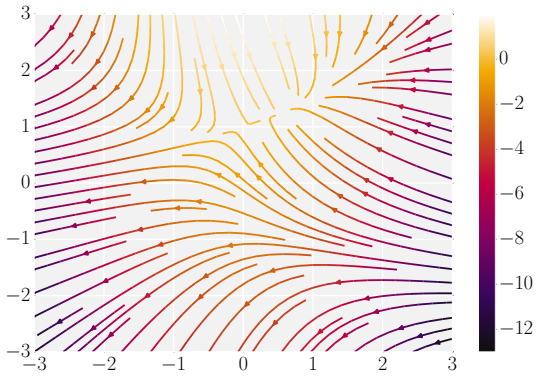

inferno

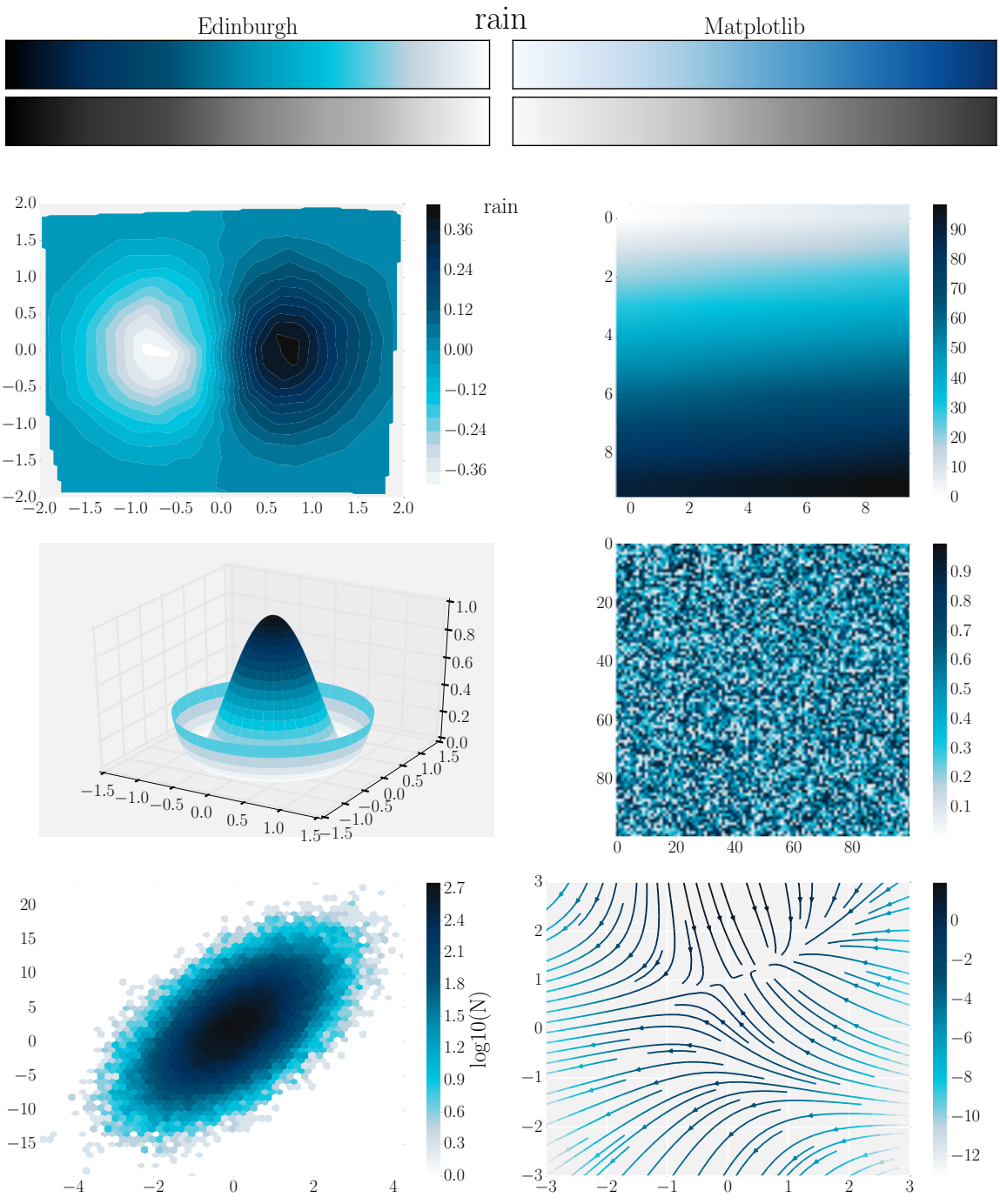


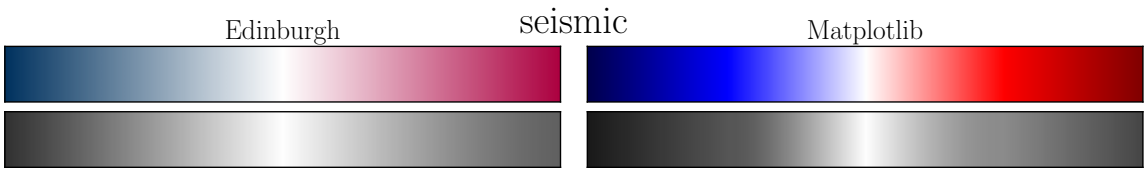


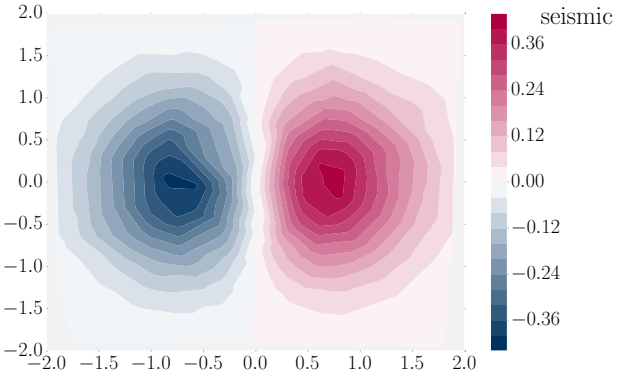


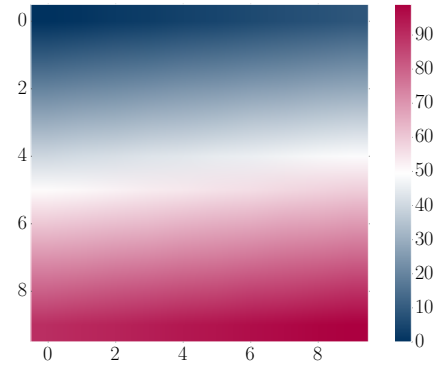

log10(N)

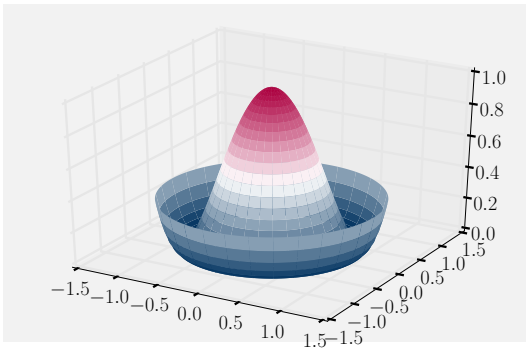


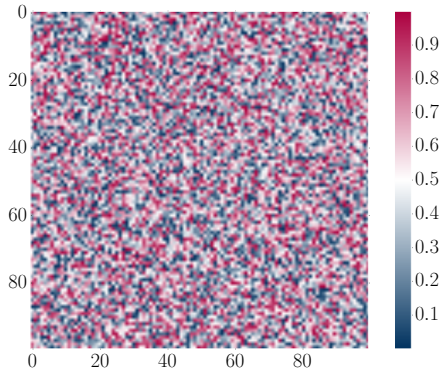

rain
Edinburgh
Matplotlib
rain
log10(N)

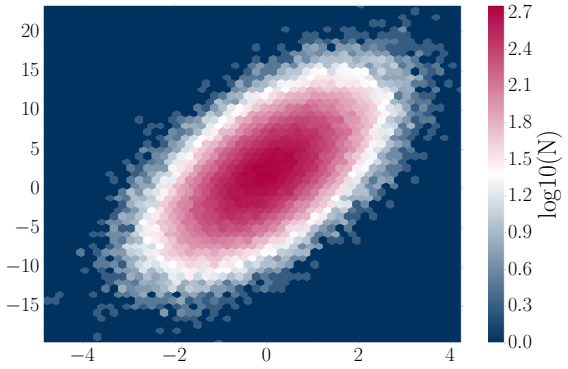

seismic
Edinburgh
Matplotlib

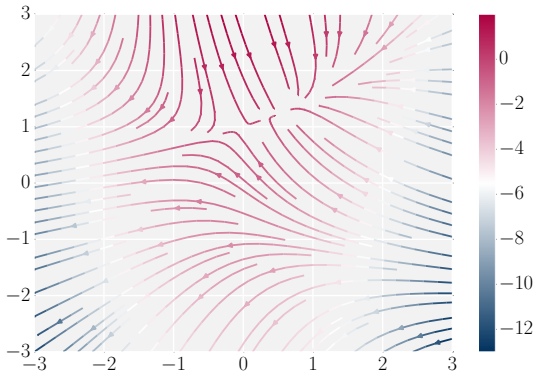

seismic

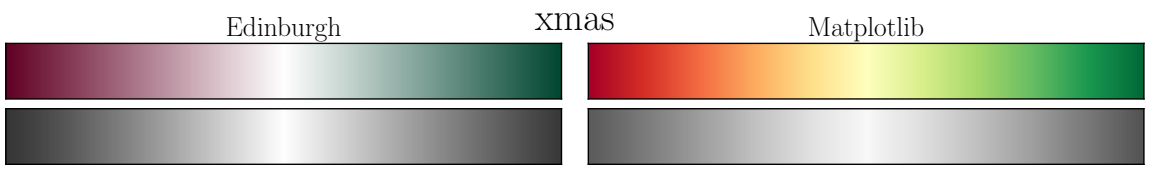


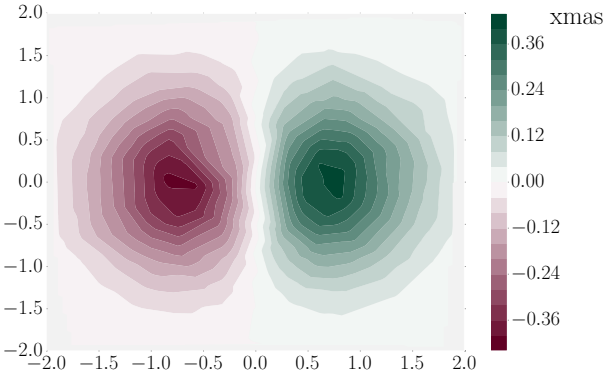


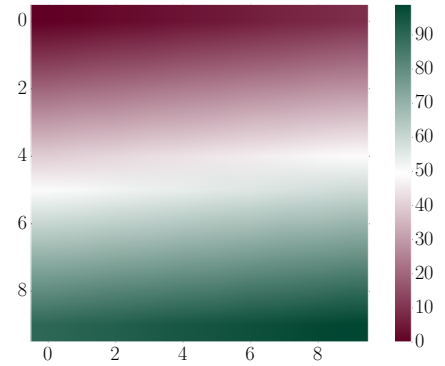


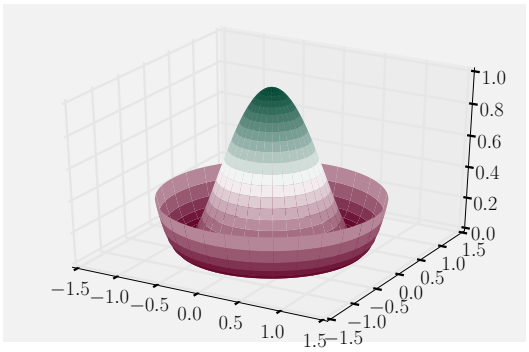

log10(N)

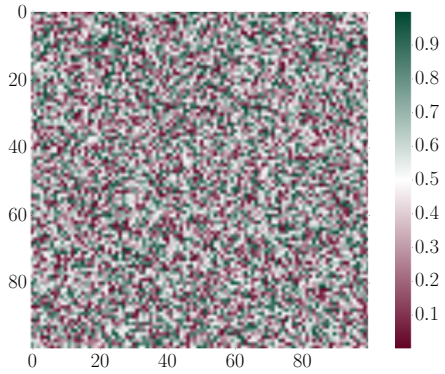


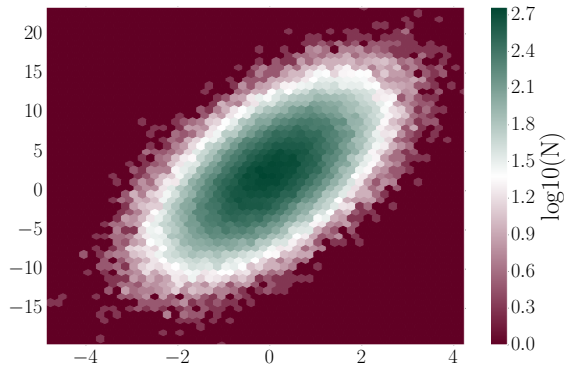

Edinburgh
xmas
Matplotlib

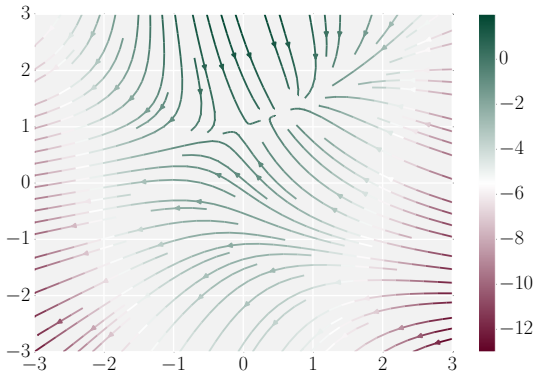

xmas

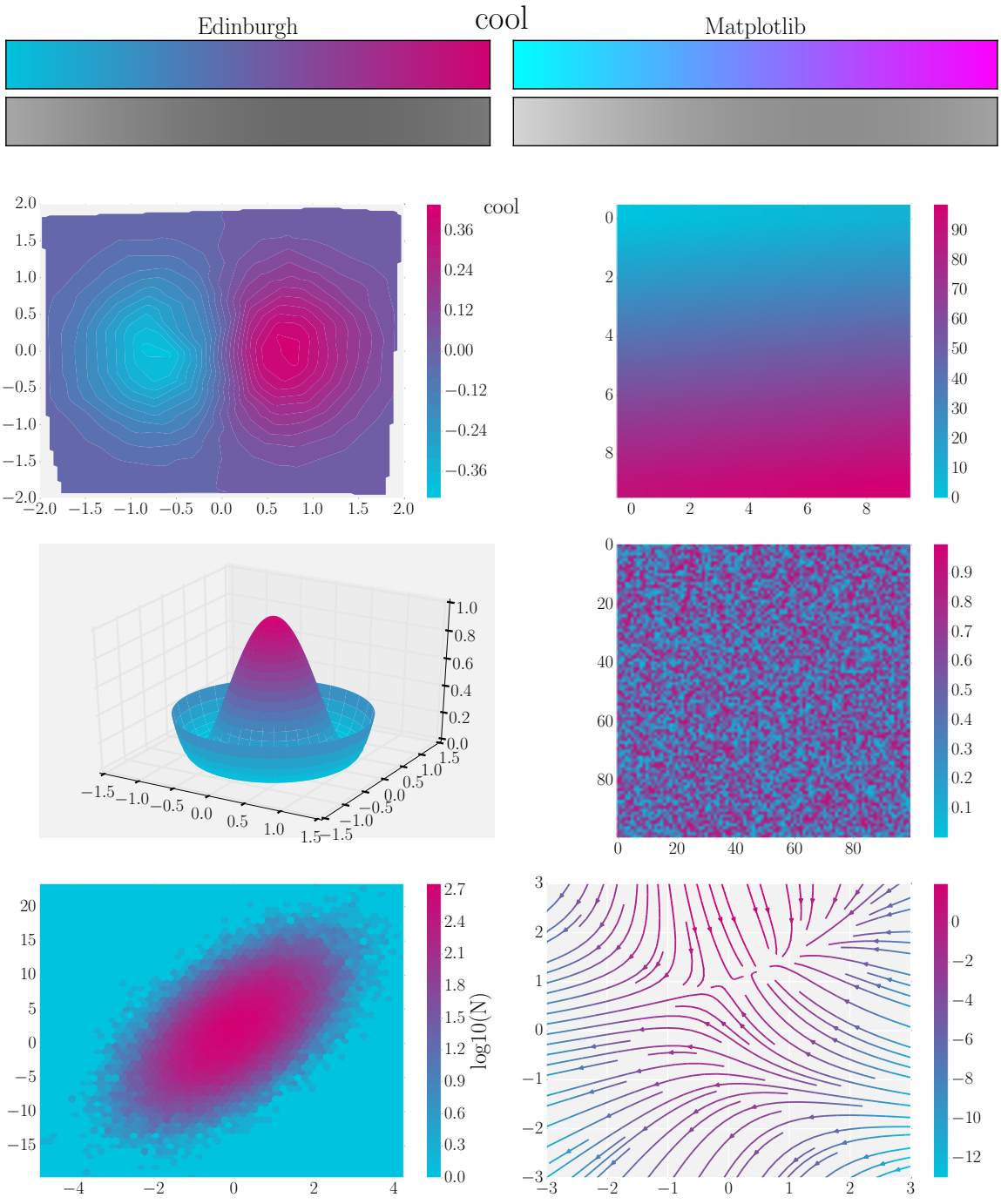


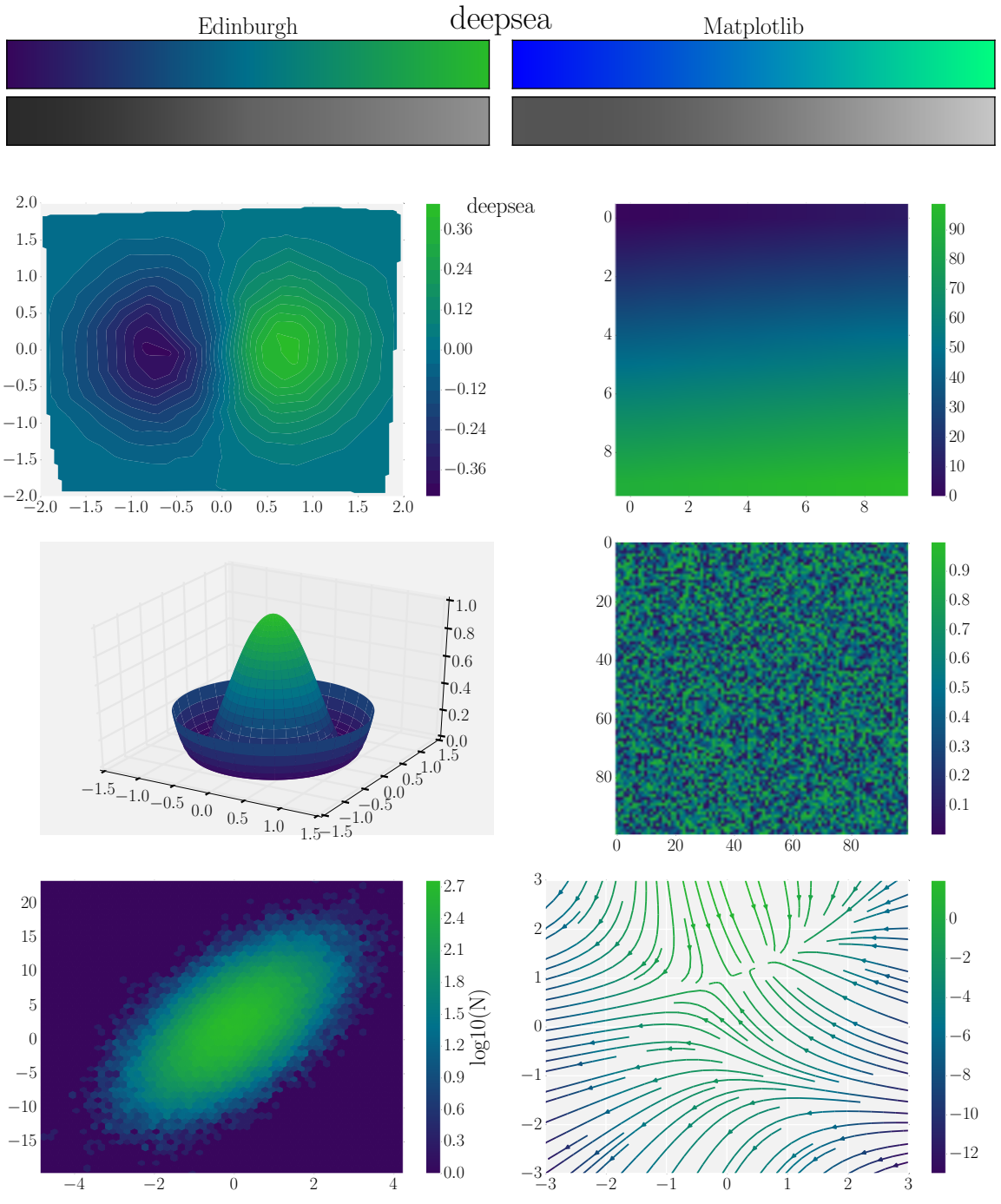





log10(N)




cool
Edinburgh
Matplotlib
cool
log10(N)


deepsea
Edinburgh
Matplotlib
deepsea
log10(N)