making `matplotlib` pretty

Luke Conaboy
A108, `luke.conaboy@nottingham.ac.uk`

I'm no expert...

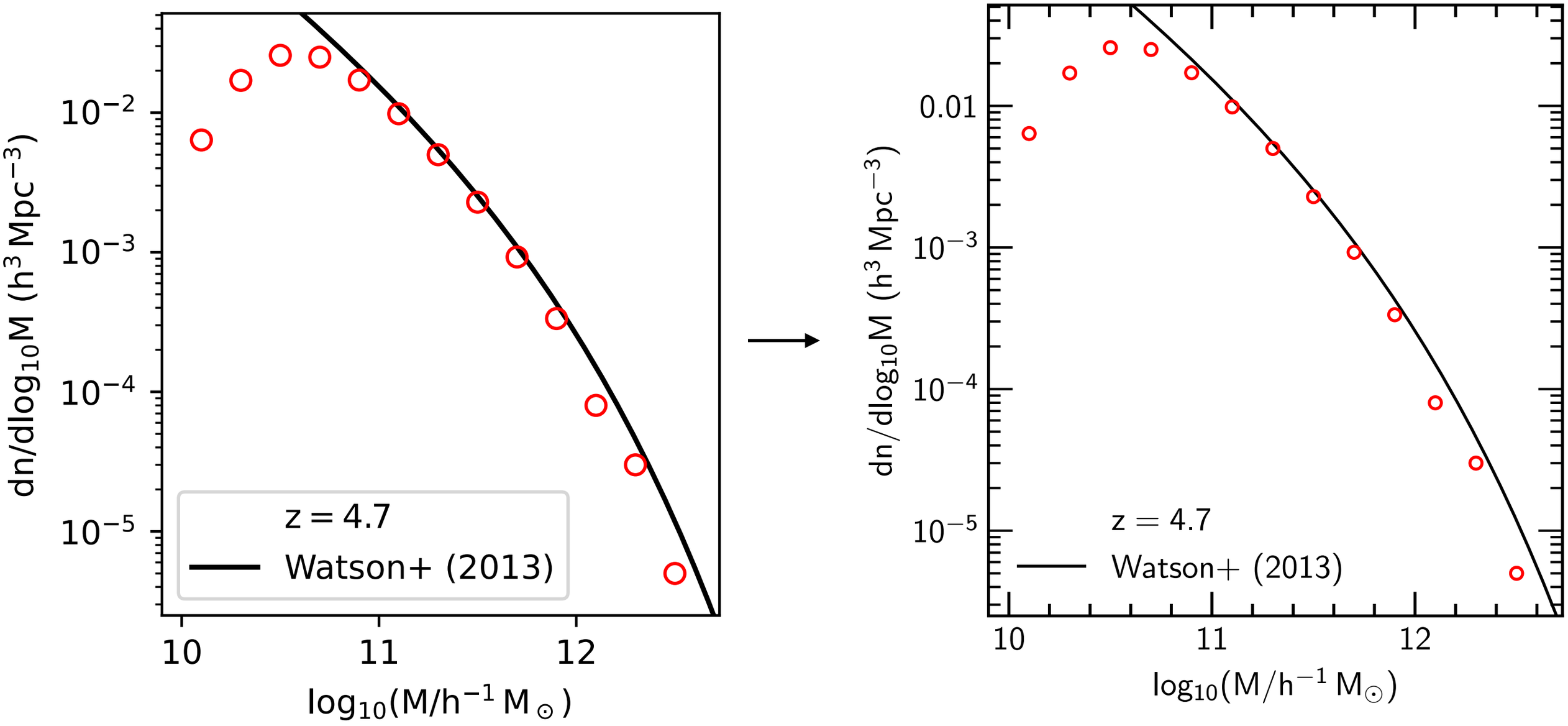
Ciaran O'Hare's great talk (+ code!) on designing plots:

https://github.com/cajohare/HowToMakeAPlot

Monica Turner's notes on changing the matplotlib defaults to look good (i.e. like old-style astronomy plots):

https://turnermoni.ca/python3.html

Monica Turner's guide



*partly subjective!

Some useful things I've learnt

- colours
- the `matplotlibrc` file
- saving as PDF
- setting your figure size directly
- using the objects
- `kwargs` (Bradley)

Some useful things I've learnt

- colours
- the `matplotlibrc` file
- saving as PDF
- setting your figure size directly
- using the objects
- `kwargs` (Bradley)

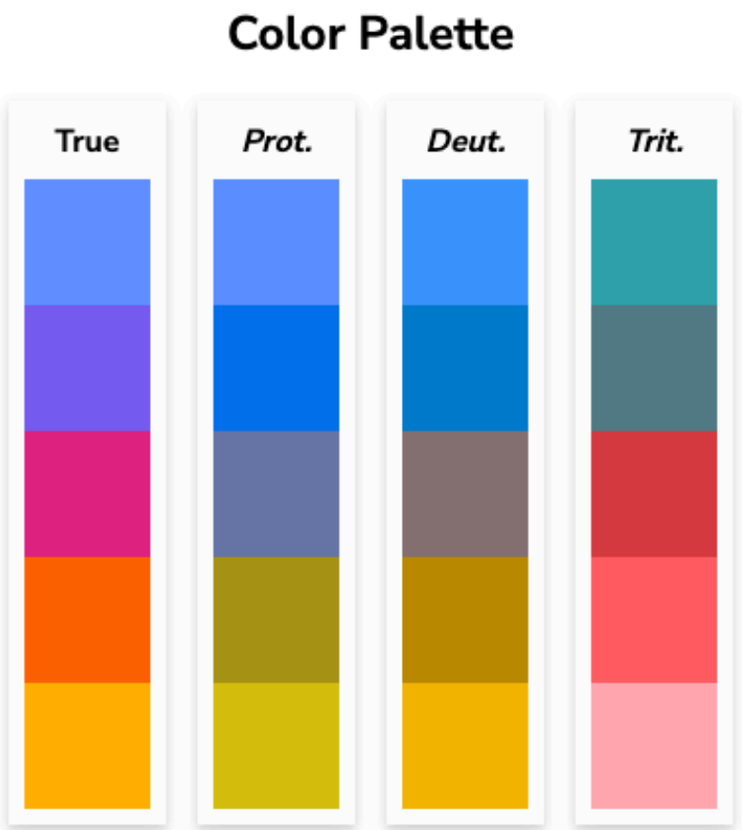
Colours

Avoid relying solely on colours to convey meaning, try using, e.g., different linestyles too

If relying on colours for lines, try using some that are distinct in greyscale (when printed) and for colourblindness:

https://davidmathlogic.com/colorblind/

https://personal.sron.nl/~pault/



Colours

Choice of colourmap for image data depends on what you're trying to show

`matplotlib` has some good and bad maps, `cmasher` has lots of good options

https://cmasher.readthedocs.io/

Some useful things I've learnt

- colours
- the `matplotlibrc` file
- saving as PDF
- setting your figure size directly
- using the objects
- `kwargs` (Bradley)

matplotlibrc

Instead of setting plotting styles for each script using `rcParams`, e.g.:

```
import matplotlib as mpl
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['lines.linestyle'] = '--'
```

we can modify the `matplotlibrc` file, which controls the default behaviour for *all scripts*!

matplotlibrc

You can have multiple `matplotlibrc` files, and they are read in the following order:

1. current working directory
2. your home, `~/.matplotlib/` (mac) `~/.config/matplotlib/` (linux)
3. the python install directory (DON'T TOUCH THIS ONE)

Setting up the `matplotlibrc` in your home (2) will apply to all your plots, and in the current working directory (3) just to the project you're working on

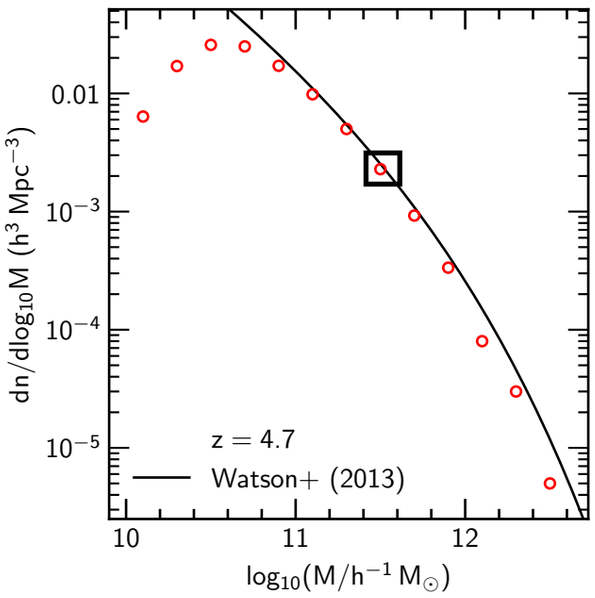
Some useful things I've learnt

- colours
- the `matplotlibrc` file
- saving as PDF
- setting your figure size directly
- using the objects
- `kwargs` (Bradley)

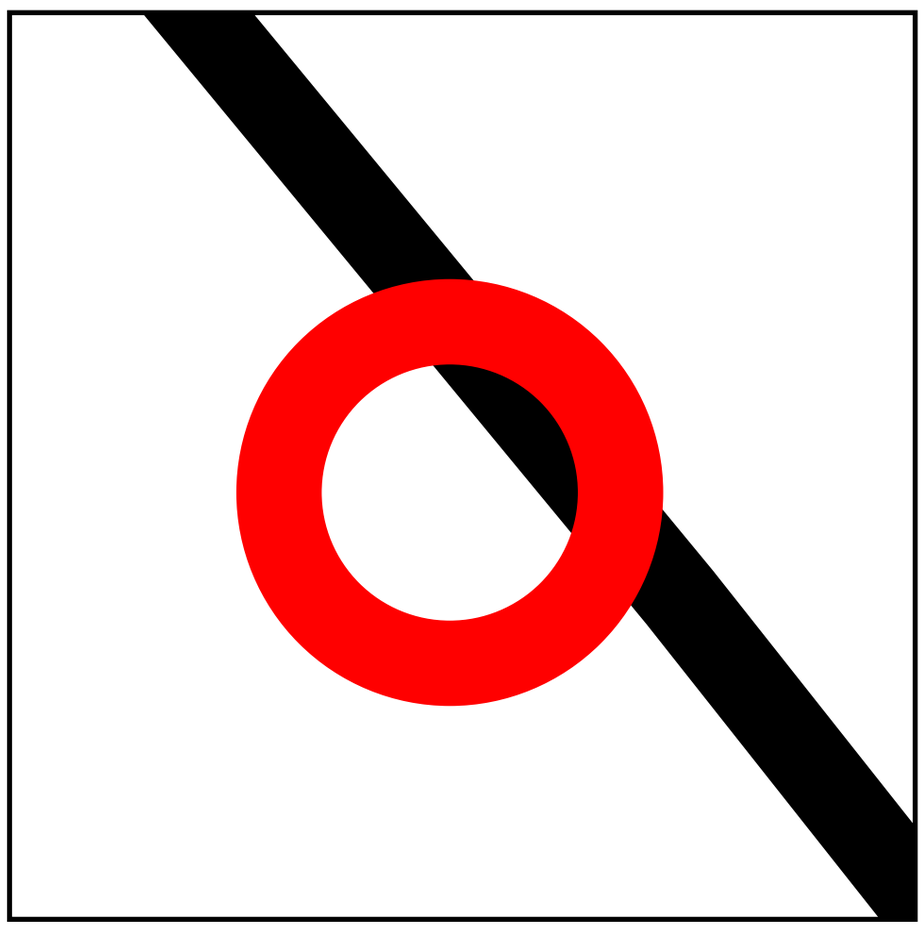
PDFs

Save as a PDF when you can*

PDFs are vector graphics, PNGs are raster graphics



`fig.savefig('my file.pdf')`



`fig.savefig('my file.png', dpi=400)`

PDFs

*except when:

- plotting a large number of lines or points, as this makes the PDF large and slow to load

- plotting images (e.g. `imshow`), these are saved as rasters and so you need to specify the dpi, even when saving as a .pdf:

```
fig.savefig('my_file.pdf', dpi=400)
```

Some useful things I've learnt

- colours
- the `matplotlibrc` file
- saving as PDF
- setting your figure size directly
- using the objects
- `kwargs` (Bradley)

Figure size

Set the figure size according to where the figures are going. A single column in MNRAS is 10/3 inches, so

```
fig, ax = plt.subplots(figsize=(3.3, 3.3))
```

will give you a square figure perfectly sized for a single-column plot in MNRAS, and there's no surprise rescaling of plot sizes!

Some useful things I've learnt

- colours
- the `matplotlibrc` file
- saving as PDF
- setting your figure size directly
- using the objects
- `kwargs` (Bradley)

Objects

Working with the `Figure` and `Axes` objects (`fig` and `ax` here) gives you more control over your plotting, and can be accessed directly at figure creation with

```
fig, ax = plt.subplots(figsize=(3.3, 3.3))
```

Avoids calls like `plt.gca()` to get to your axes, or having to plot things in a specific order (e.g. in a loop)

See the documentation for the methods of each:

`Figure:` https://matplotlib.org/stable/api/figure_api.html

`Axes:` https://matplotlib.org/stable/api/axes_api.html