Making pretty plots in python

Eric C. J. Oliver

Institute for Marine and Antarctic Studies
University of Tasmania, Hobart, Australia

translatingnatureintoknowledge

Getting started with python

- Python comes pre-installed on many systems by default (most linux flavours, OS X – not windows, *surprise surprise*), just open a terminal and type `python`
- However, to do science-y, plot-y things in python we require more than the basic install – we require some extra modules, e.g.
  - `numpy` and `scipy`
  - `matplotlib` and `basemap`
- So you can either install these packages yourself (can be painful) or install another python distribution which includes them all be default (such as those from www.enthought.com)
- However, you will probably end up learning how to install modules yourself anyway (`NetCDF`)
- Also, I recommend using the `ipython` shell, instead of the basic python shell, as it adds command history, autocompletion, etc...

Getting started with python

- `numpy` and `scipy` (usually loaded as `np` and `sp`) will be your "bread and butter" for math, stats, and science computation
  - www.numpy.org, www.scipy.org
- `matplotlib` and in particular `pyplot` (usually loaded as `plt`) will contain most of the basic plotting functions you will need, and was developed to create a MATLAB-like set of plotting tools
  - matplotlib.org
- `basemap` (usually loaded as `bm`) will allow to you plot 2D data on a map, with functionality for map projections, coastlines, etc...
  - matplotlib.org/basemap/

Getting started with python

- A typical start of my python session, which I run from the terminal:

```
> import numpy as np
> import scipy as sp
> from matplotlib import pyplot as plt
> import mpl_toolkits.basemap as bm
```

  This is like “adding to your path” in MATLAB except you need to tell python every time which modules you want to use in that session

- Then the commands you will use are part of one of the modules you loaded, e.g.,

```
> a = 5
> np.sqrt(a)
2.2360679774997898
```

Basic plotting commands

Command	Function
`plt.figure(figsize=(X,Y))`	Open new figure window, size X by Y
`plt.plot(x,y)`	Plot y versus x

`plt.xlabel(string)`
`plt.ylabel(string)`
`plt.title(string)`
	Label x-axis, y-axis and plot title

`plt.xlim(x1, x2)` `plt.ylim(y1, y2)`	Set bounds on x and y axes
`plt.grid()`	Add grid lines to plot
`plt.subplot(r,c,i)`	Create new axes as part of a r x c matrix of subplots, with 1 ≤ i ≤ (rc)

`plt.contour(x,y,z,`
`levels=ZCONT,`
`cmap=plt.cm.CBAR)`
	Contour plot of z(x,y) with contours at values of z specifed by vector ZCONT and with colorbar CBAR

`plt.contourf(x,y,z,`
`levels=ZCONT,`
`cmap=plt.cm.CBAR)`
	As above but gives filled contours

`plt.quiver(x,y,u,v)`	Quiver/velocity plot of u(x,y), v(x,y)
`plt.colorbar()`	Add colorbar to plot
`plt.clim(z1, z2)`	Set bounds on colorbar/map
`plt.clf()`	Clear current figure

Simple plot of *x*-*y* data

- Let's say we have a set of four time series in the variable `T_CTRL` and another four in the variable `T_A1B`, with time values given in the variable `time`

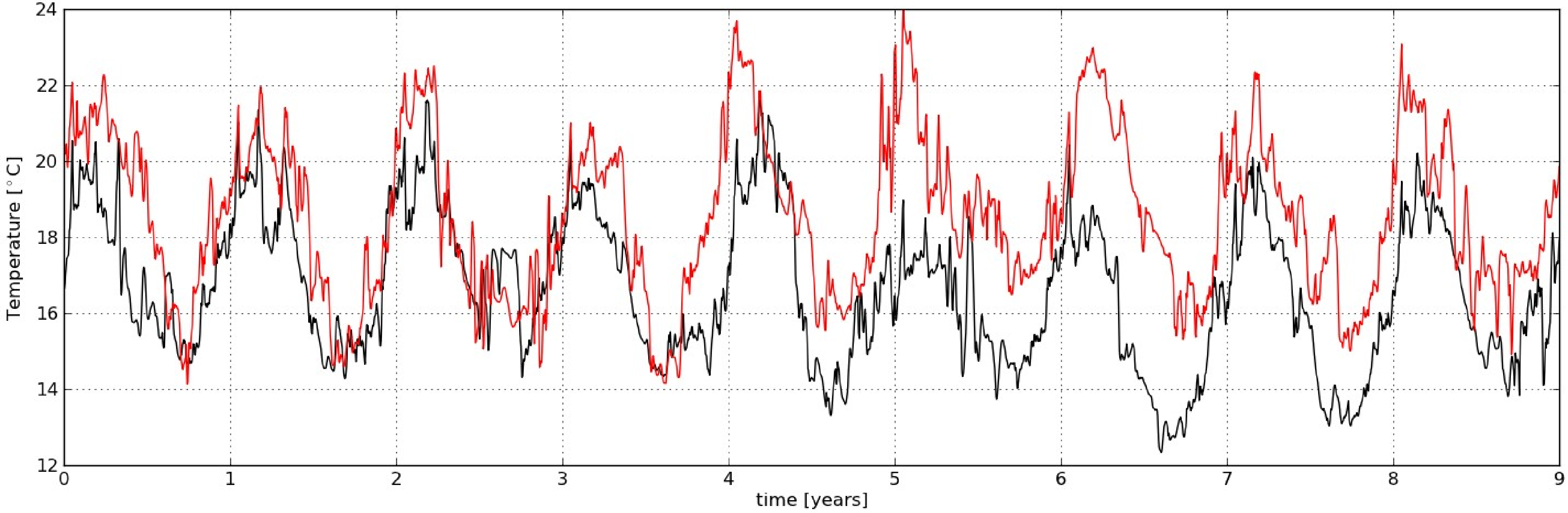
```
> T_CTRL.shape
(3285, 4)
> time.shape
(3285,)
```

- So, let's plot the first time series' from `T_CTRL` and `T_A1B`:

```
plt.figure()
plt.plot(time, T_CTRL[:,1], 'k-')
plt.plot(time, T_A1B[:,1], 'r-')
plt.ylabel(r'Temperature [$^\circ$C]')
plt.xlabel('time [years]')
plt.grid()
```

- Note that adding `r` before a character string tells python to interpret LaTeX commands in that string

Simple plot of *x*-*y* data



- And you can easily save your plot as a raster image or as a vector imate using `plt.savefig`:

```
plt.savefig('temp_ts.png') # save as raster image
plt.savefig('temp_ts.pdf') # save as vector image
```

Basic mapping commands

Command	Function
`proj = bm.Basemap(projection='merc', llcrnrlat=lat1, llcrnrlon=lon1, urcrnrlat=lat2, urcrnrlon=lon2, resolution='i')`	Create `proj` object for Mercator (`'merc'`) projection with lat and lon bounds given by (`lat1`,`lat2`) and (`lon1`,`lon2`) and with intermediate resolution (`'i'`) coastline
`proj.drawcoastlines()`	Draw coastlines, obviously
`proj.fillcontinents(color='white')`	Fill continents inside coastlines with colour
`proj.drawparallels(LATS, labels=[True,False,False,False])` `proj.drawmeridians(LONS, labels=[False,False,False,True])`	Draw parallells and meridians (ticks and grid) specified by vectors `LATS` and `LONS`, and `labels` specifies where to place label (L,R,T,B)
`llon, llat = np.meshgrid(lon, lat)`	Put `lat`/`lon` vectors onto a 2D grid
`lonproj, latproj = proj(llon, llat)`	Transform (`llat`,`llon`) from decimal degrees to map projection coordinates
Normal plotting commands now work, provided you use the transformed coordinates, e.g.,	
`plt.contourf(lonproj, latproj, z)`	Filled contour plot of z(lon, lat)

Simple contour map of *z*(*x*,*y*) data

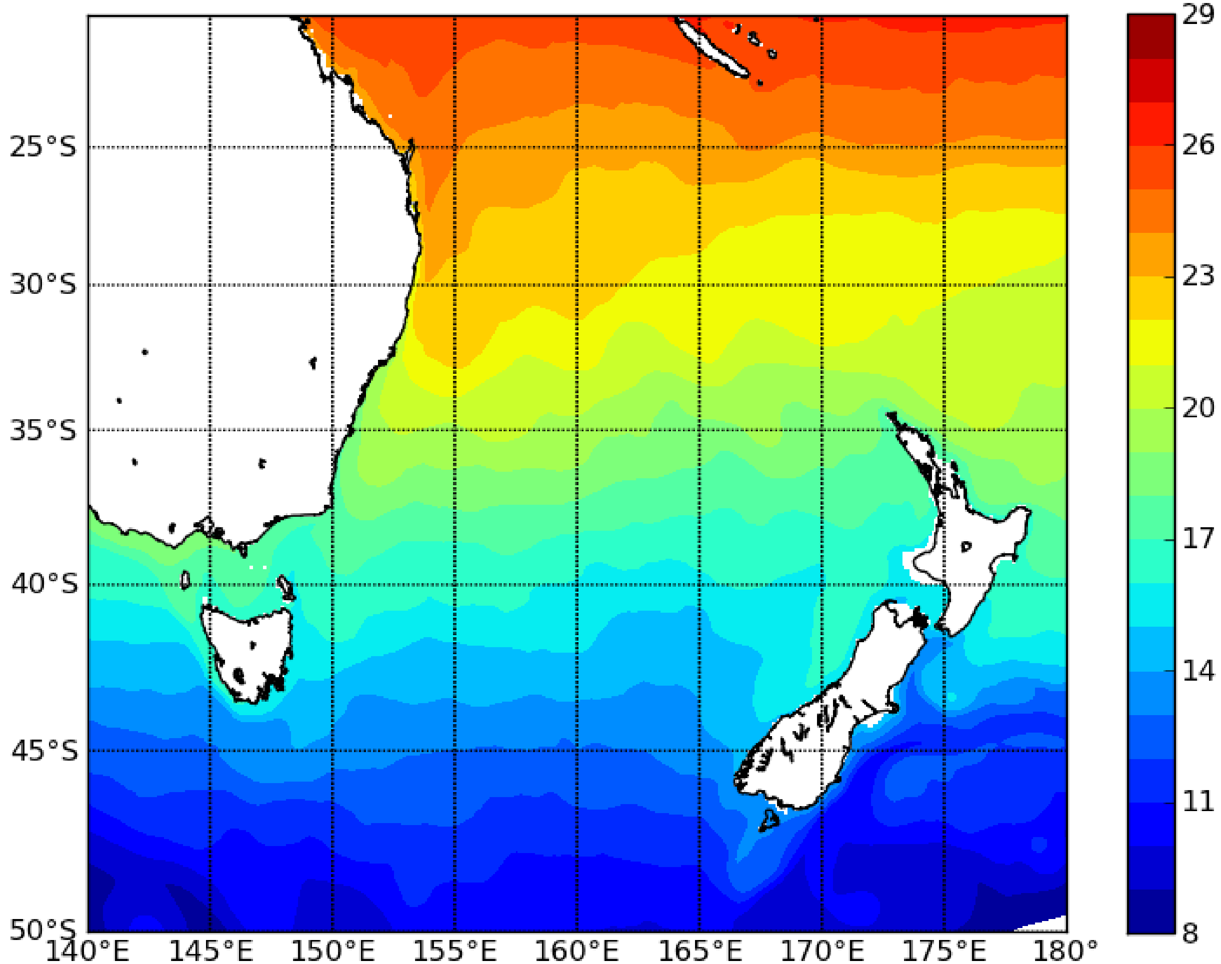
- Let's say we have variables `T_CTRL` and `T_A1B` which are maps of mean SST at coordinates `lon` and `lat`

```
> T_CTRL.shape
(701, 901)
> lon.shape
(901,)
> lat.shape
(701,)
```

- So, let's define our projection and map it up:

```
> plt.figure()
> proj = bm.Basemap(projection='merc', llcrnrlat=-50, llcrnrlon=140,
urcrnrlat=-20, urcrnrlon=180, resolution='i')
> proj.drawcoastlines()
> proj.drawparallels([-50,-45,-40,-35,-30,-25,-20],
labels=[True,False,False,False])
> proj.drawmeridians(range(140,180+1,5), labels=[False,False,False,True])
> llon, llat = np.meshgrid(lon, lat)
> lonproj, latproj = proj(lon, lat)
> plt.contourf(lonproj, latproj, T_CTRL, levels=np.arange(8,30,1))
> plt.colorbar()
```

Simple contour map of *z*(*x*,*y*) data



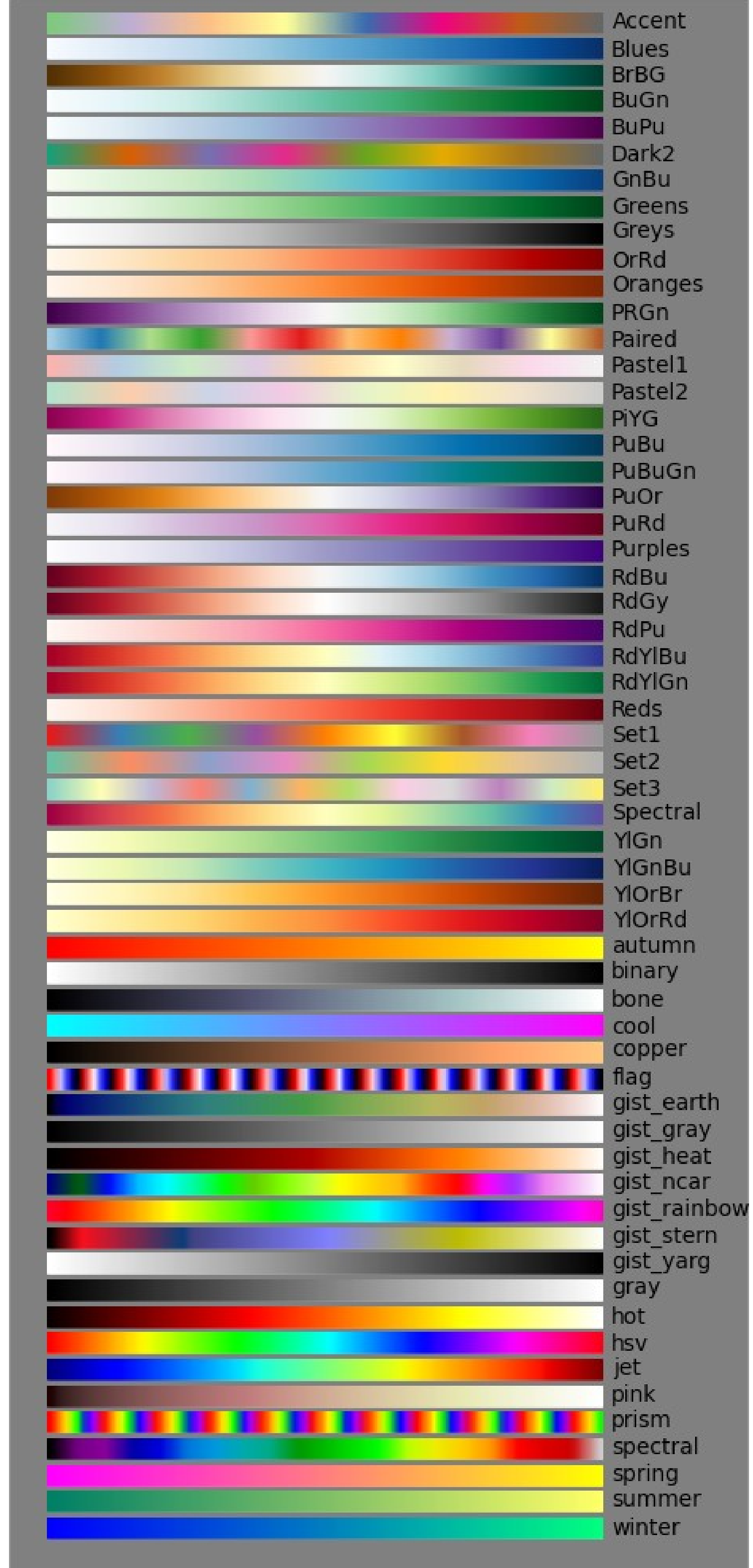
- Can play with the colormap by specifying CMAP:

```
> plt.contourf(..., cmap=plt.cm.CMAP)
```

Colormaps available in matplotlib by default.

This list is not exhaustive... google is your friend

Also, as a further demonstration of python's awesomeness: you can append `_r` to ANY colormap to flip the colour order!



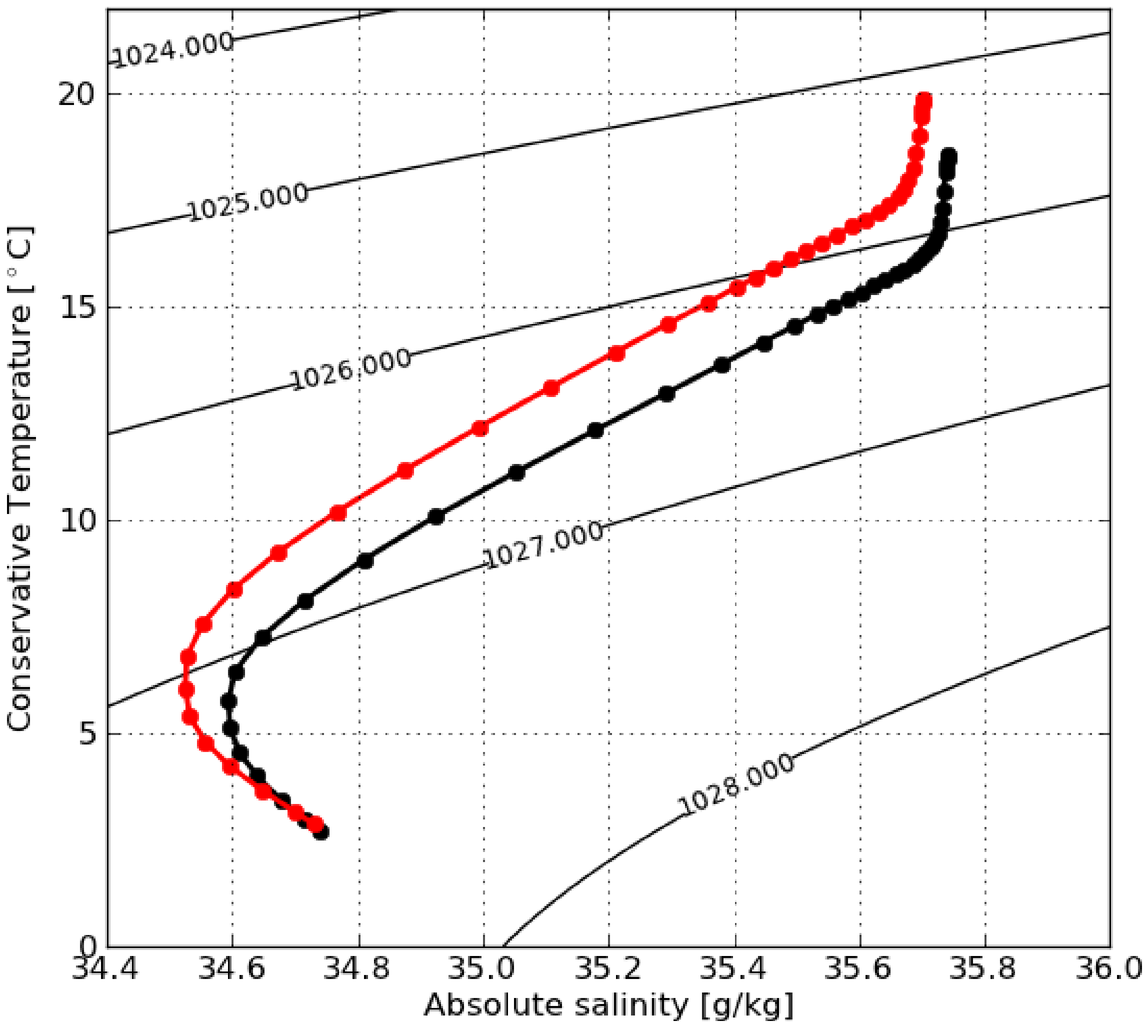
wiki.scipy.org/Cookbook/Matplotlib/Show_colormaps

T-*S* diagrams

- In order to do a lot of oceanography calculations we will need functions to convert between depth/pressure, in-situ/conservative/potential temperature, practical/absolute salinity, etc...
- The python module `gsw` (pypi.python.org/pypi/gsw/) is an implementation of the Thermodynamic Equation of Seawater - 2010 (www.teos-10.org) and gives you access to these function
- After building and installing simply load the module in python:

```
> import gsw
```

T-*S* diagrams



- ...go to demo for details...

The End

- These are just the basics, more advanced techniques are out there
- There is a TON of information available
  - scipy.org, numpy.org - tutorials
  - matplotlib.org - tutorials
  - Online forums usually answer any question... google around
  - Ask me, or other python-ers (pythonians?)
- Final comments
  - I have found NO plotting tool provides “really beautiful” plots by default and some post-processing (e.g., Adobe Illustrator, Inkscape) is usually necessary
  - if there is enough interest I can give a talk on how to use inkscape for touching up figures and also for making diagrams and posters