Modelling and data analysis 'Winter School'

Nick Golledge[1],
Liz Keller[1,2],
Alex Gossart[1],
Alena Malyarenko[3],
Angela Bahamondes-Dominguez[3],
Mario Krapp[2],
Dan Lowry[2],
Stefan Jendersie[1]

[1] Antarctic Research Centre, Victoria University of Wellington, New Zealand
[2] GNS Science, Lower Hutt, New Zealand
[3] NIWA, Wellington, New Zealand

Antarctic Science Platform
National Modelling Hub

MINISTRY OF BUSINESS, INNOVATION & EMPLOYMENT
HĪKINA WHAKATUTUKI

NIWA
Taihoro Nukurangi

1 | Welcome & Introduction

Day 1

10:00	Arrival & welcome	Nick
10:15	Introduction to programming Navigating the command line environment, scripting vs programming, pros & cons of various languages	Nick
11:30	Introduction to models Climate model basics: components, types of models, internal variability. CMIP overview, climate sensitivity	Liz & Dan
13:00	Lunch	
14:00	Spatial data – lecture Understanding gridded data, map projections, data analysis and manipulations, masking, extracting vertical / horizontal sections	Alex & Alena
15:30	Coffee	
15:45	Spatial data – tutorial	Alex & Alena
17:00	Wrap-up	

Day 2

09:00	Time-series data – lecture Principal component / empirical orthogonal function analysis, calculation of correlations, anomalies, detrending	Mario
10:30	Coffee	
10:45	Time-series data – tutorial	Mario
12:15	Lunch	
13:15	Document preparation in LaTeX Learn the basics, write equations, insert figures, create your own tables, insert references	Angela
14:45	Afternoon tea	
15:00	Work Structure & Version control Defining a workflow, handling 'big data', version control for scripts/documents, best practice guidelines	Stefan

- The **aim** of the Winter School is that, by the end of the two days, participants will be able to find and download (climate model) data of interest, use simple scripts to process, analyse, and plot those data, integrate these outputs into a typeset document, and use version control software to keep track of changes.

- We will use ***Python*** for the majority of the work but will incoporate examples from other languages if necessary. We'll introduce you to packages like LaTeX and tools such as *github*.

- This workshop is only intended to provide an **introduction** to working in a command-line environment, and exposure to some of the functionality available in this realm. It is not intended to be a complete course on programming, modelling, or data analysis ;-)

- The **aim** of the Winter School is that, by the end of the two days, participants will be able to find and download (climate model) data of interest, use simple scripts to process, analyse, and plot those data, integrate these outputs into a typeset document, and use version control software to keep track of changes.

- We will use ***Python*** for the majority of the work but will incoporate examples from other languages if necessary. We'll introduce you to packages like LaTeX and tools such as *github*.

- This workshop is only intended to provide an **introduction** to working in a command-line environment, and exposure to some of the functionality available in this realm. It is not intended to be a complete course on programming, modelling, or data analysis ;-)

- The **aim** of the Winter School is that, by the end of the two days, participants will be able to find and download (climate model) data of interest, use simple scripts to process, analyse, and plot those data, integrate these outputs into a typeset document, and use version control software to keep track of changes.

- We will use ***Python*** for the majority of the work but will incoporate examples from other languages if necessary. We'll introduce you to packages like LaTeX and tools such as *github*.

- This workshop is only intended to provide an **introduction** to working in a command-line environment, and exposure to some of the functionality available in this realm. It is not intended to be a complete course on programming, modelling, or data analysis ;-)

2 | Command-line basics (*nix)

Basic commands

command	example	description
ls	`ls -ltrh`	list directory contents (in long format, newest last)
cd	`cd ../mydir/mysubdir`	change directory (up one level, down two)
rm	`rm delete-this.txt and-all-these.*`	remove file(s)
mv	`mv rename-this.txt to-this.txt`	move (rename) file(s)
mkdir	`mkdir ./new-directory`	make a new (empty) directory
cp	`cp this.txt ./new-dir/to-this.txt`	copy file (possibly to new location)

Linux c-line tools

tool	example	description
pwd	`pwd`	Find out what your current personal working directory is
sed	`sed -e 's/a/b/g'`	stream editor, swap 'a' for 'b'
awk	`awk '{print $2, $3}'`	print fields 2 & 3 from file/stream

Other packages & utilities

package	example	description
pdflatex	`pdflatex myfile.tex`	compile LaTeX document
git	`git clone golledni/WinterSchool`	Make a local copy of a github repository

- We can combine many simple commands, tools, and utilities to achieve more complex tasks

```
pwd
/home/golledni/MEGA/Work/AntSciPlat/WinterSchool

pwd | sed -e 's/\// /g' | awk '{print "Today my",$1," is the",$NF}'
Today my home is the WinterSchool
```

- But to do anything more complex than simple pipes we probably want to write a script file to contain our sequence of commands:

```
#!/bin/sh

lastupdated=`head -n 1 new_papers.txt`

echo "Last updated " $lastupdated

now=$(date +%F)

echo $now > new_papers.txt

# get list of directories to loop through

list=`ls -l | grep ^d | awk '{print $9}'`

# find NEW papers in each of those directories

echo "\nNEW PAPERS:\n" >> new_papers.txt

for dir in $list ; do

        echo "\n********** "$dir" **********\n" >> new_papers.txt

        find ./$dir -type f -newermt $lastupdated -print | awk -F"/" '{print $3}' | sed -e 's/.pdf/}
/g' -e 's/_/ /g' -e 's/^/\\subsubsection{/' >> new_papers.txt

done
```

- Often we want to apply a test, or series of tests, to the data we're processing, and do different things with the data depending of the results of those tests
- Control structures are what help us achieve this, and are fundamental to all languages
- The two most common generic structures are
  - `if` statements, and
  - `for` or `do` loops

`if` statement:

```
i=0
if [ $i -ge 1 ]
then
   echo "i = $i"
else
   echo "i < 1"
fi
```

`do` loop:

```
i=0
imax=10
while [ $i -le imax ] ; do
   echo "i = $i"
   i=`expr $i + 1`
done
```

Bash:

```
#!/bin/sh

echo "Hello, World!"
```

Python:

```
#!/usr/bin/env Python

print "Hello, World!"
```

Julia:

```
#!/usr/bin/env Julia

print("Hello, World!")
```

Fortran 90:

```
PROGRAM HELLOWORLD

   IMPLICIT NONE
   print *, 'Hello, World!'

END
```

C++:

```
#include <iostream>

int main() {
   std::cout « "Hello, World!";
   return 0;
}
```

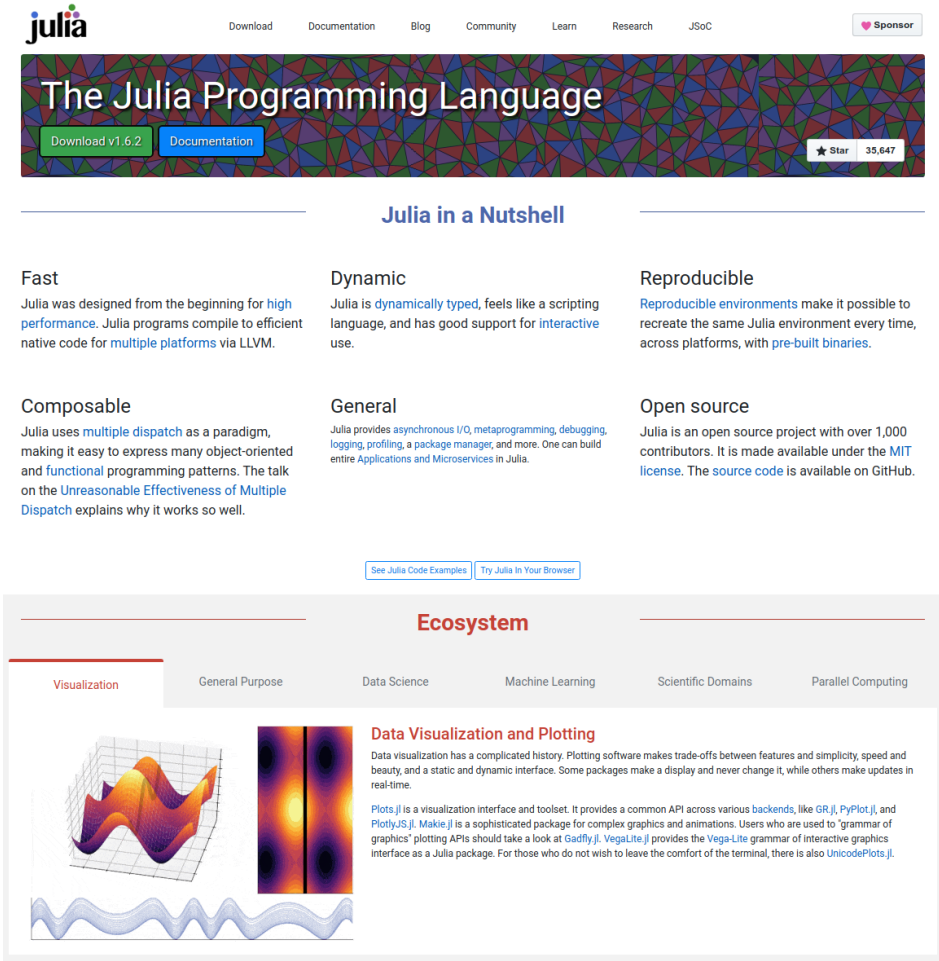
2 | Interpreted vs. compiled languages

- Compiled languages require a *compiler* to convert user code into machine code
- Typically they create a platform-dependent binary (executable) file
- If the code doesn't change, the binary can be run again and again
- Once compiled, programs using these languages are typically very fast to run

- Interpreted languages read and execute user code line-by-line
- No separate compilation step is required, so programs are platform-*in*dependent
- But, interpretation has to happen every time the code is run
- As a result, this kind of code is typically slow to run

Just-in-time (JIT) compilation:

- Some modern languages like **Julia** use the JIT (or dynamic compilation) approach
- With JIT, compilation of relevant code occurs at runtime
- If same packages / modules are called in subsequent runs, no re-compilation is necessary
- This approach combines the best aspects of interpreted and compiled languages



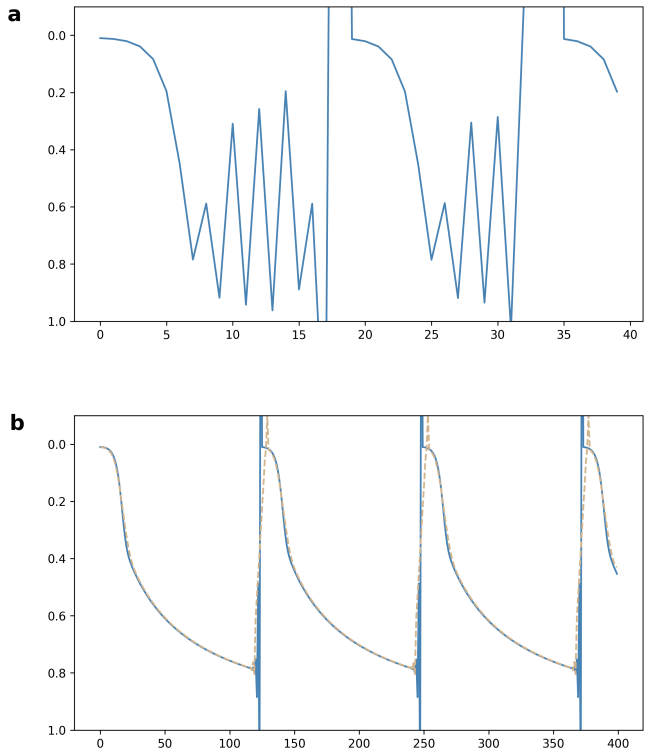
2 | Speeding things up with functions

- A function is a block of code that does a specific job
- Because it sits outside the main code sequence it only needs to be compiled and optimized once, even if it is used repeatedly
- This makes your code faster :-)

```
function movingaverage(g)
    n = 5 # number of points
    z = [i < n ? mean(g[begin:i]) : mean(g[i-n+1:i]) for i in 1:length(g)]

    return z
end

function makenorm(x)
    range = maximum(x) - minimum(x)
    y = (x .- minimum(x)) ./ range

    return y
end


function slope(x)
    grad = zeros(length(x))
    grad[2:end-1] = x[3:end] - x[1:end-2]

    return grad
end
```

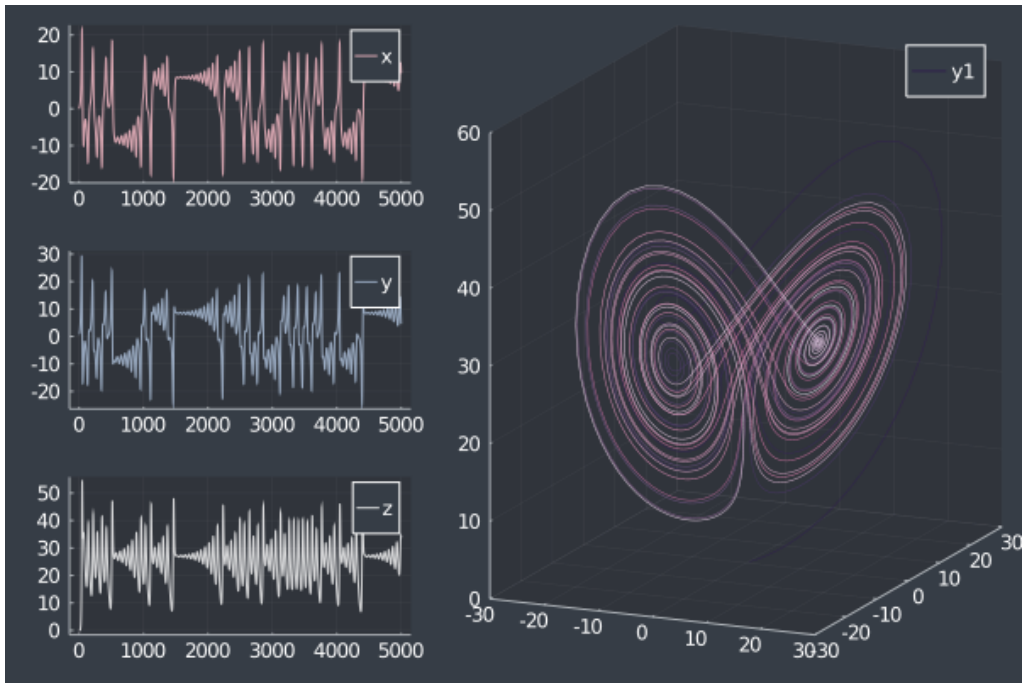
```
function readdata(reg, var, dT, dOT, type)
    regi = string(reg)
    fle = "./data/REG"*regi*"-"*dT*"-"*dOT*".txt"
    data = readdlm(fle)
    time = data[:,1]

    if var == "areagr" ; col = 2 ; end
    if var == "areafl" ; col = 3 ; end
    if var == "volgr" ; col = 4 ; end
    if var == "volfl" ; col = 5 ; end
    if var == "smb" ; col = 6 ; end
    if var == "bmb" ; col = 7 ; end
    if var == "elev" ; col = 8 ; end
    if var == "taud" ; col = 9 ; end
    if var == "btemp" ; col = 10 ; end
    if var == "bvel" ; col = 11 ; end
    if var == "svel" ; col = 12 ; end

    dat = data[:,col]

    if type == "abz"
        return movingaverage(dat)
    elseif type == "grd"
        return movingaverage(slope(dat))
    elseif type == "nrm"
        return makenorm(movingaverage(dat))
    elseif type == "nrmgrd"
        return makenorm(movingaverage(slope(dat)))
    end

end
```

- Usually, a numerical model consists of a set of calculations that are repeated
- Typically, each repetition of the solution involves a step forward in time
- A spatially *discretized* model may use an *implicit* or *explicit* time step
- Numerical solutions are prone to error (compared to an analytical solution)
- Accumulated error tends to produce instability & model crash
- Usual culprits are fluxes getting too great, or time steps being too long



- Often the equations we are trying to solve are differential equations, i.e. they describe a quantity that changes through time

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta z$$



Lorenz equations (of atmospheric convection)

2 | Numerical modelling: epidemics

A good example of a system that changes through time with no inherently predictable[1] outcome is the spread of an epidemic:

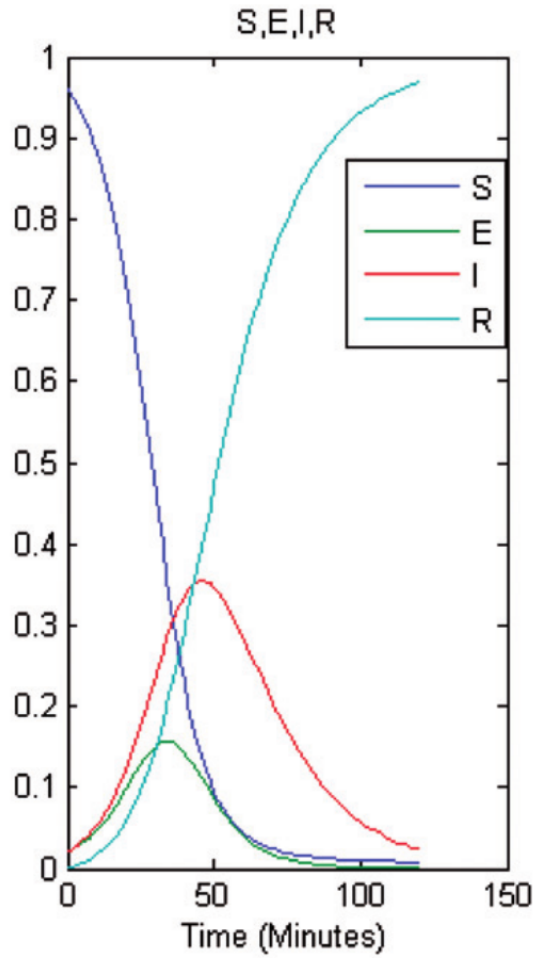
- We can define differential equations for different components of the susceptible population
  - Susceptible
  - Exposed
  - Infected
  - Recovered

$$\mathrm{d}S/\mathrm{d}t = -kI(t)S(t)$$

$$\mathrm{d}E/\mathrm{d}t = kI(t)S(t) - \varepsilon E(t)$$

$$\mathrm{d}I/\mathrm{d}t = \varepsilon E(t) - \eta I(t)$$

$$\mathrm{d}R/\mathrm{d}t = \eta I(t)$$



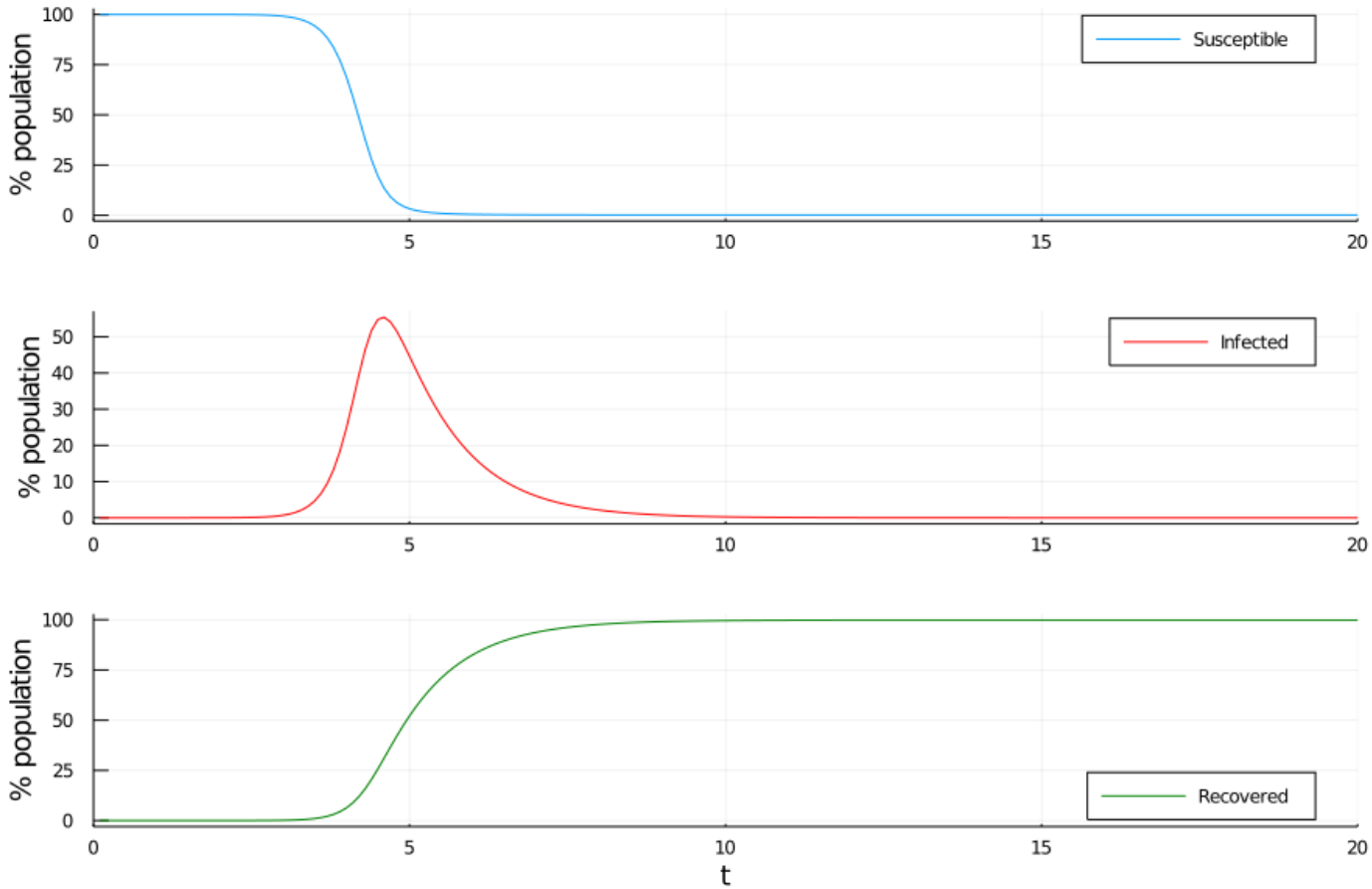
(Bilge et al., 2012)

2 | Numerical modelling: epidemics

```
#!/usr/bin/env julia

using StatsPlots, StatsPlots.PlotMeasures
using Plots; gr(size=(900, 600), bg=:white,
                xtickfontsize=8, ytickfontsize=8,
                xguidefontsize=12, yguidefontsize=12,
                bottom_margin=5mm, left_margin=5mm)
using DelimitedFiles
using Statistics
using Interpolations
using LaTeXStrings

n= 250

pop = 5e6

S = zeros(n)
E = zeros(n)
I = zeros(n)
R = zeros(n)

S[:] .= 1
E[:] .= 0
I[:] .= 1e-7
R[:] .= 0

beta = 5.7     # transmissibility?
eta = # 1.5
nabla = 4.5   # recovery time?

dt = 0.1
t = collect(1:n) .* dt
tmax = n * dt

dSdt = 0.
dEdt = 0.
dIdt = 0.
dRdt = 0.

xmax = 20
```

```
# anim = @animate for i = 2:n
for i = 2:n
#    t[i] = t[i-1] + dt

    dSdt = -beta * S[i-1] * I[i-1]
    S[i] = S[i-1] + (dSdt * dt)

    # dEdt = (beta * S[i-1] * I[i-1]) - (eta * E[i-1])
    # E[i] = E[i-1] + (dEdt * dt)

    dIdt =  (beta * S[i-1] * I[i-1]) - (nabla * I[i-1])
#    dIdt = (eta * E[i-1]) - (nabla * I[i-1])
    I[i] = I[i-1] + (dIdt * dt)

    dRdt = nabla * I[i-1]
    R[i] = R[i-1] + (dRdt * dt)
end
# gif(anim, pwd()*"/epidemic.gif", fps = 10)
p1 = plot(t, ((S .* pop)./pop) .* 100, xlims=(0, xmax), lab="Susceptible",
            ylabel="% population")

# p2 = plot(t, ((E .* pop)./pop) .* 100, xlims=(0,xmax), color=:brown, lab="Exposed",
#             ylabel="% population")

p3 = plot(t, ((I .* pop)./pop) .* 100, xlims=(0, xmax), color=:red, lab="Infected",
            ylabel="% population")

p4 = plot(t, ((R .* pop)./pop) .* 100, xlims=(0, xmax), color=:green, lab="Recovered"
            legend=:bottomright, xlabel="t", ylabel="% population")


l = @layout [a ; b ; c]

## Make plot

plot(p1, p3, p4, layout=l)

png(pwd()*"/epidemic")
```

2 | Numerical modelling: epidemics



- Simple 1D system treats entire population as a bulk quantity
- Makes *lots* of simplifying assumptions:
  - Entirety of population are equally susceptible
  - 'Perfect' transmission occurs
  - Recovery is just a matter of (uniform) time
  - Full immunity follows
- Good for understanding evolution of a simple system, but not very realistic

Is it realistic to simulate epidemic evolution as a diffusive system?

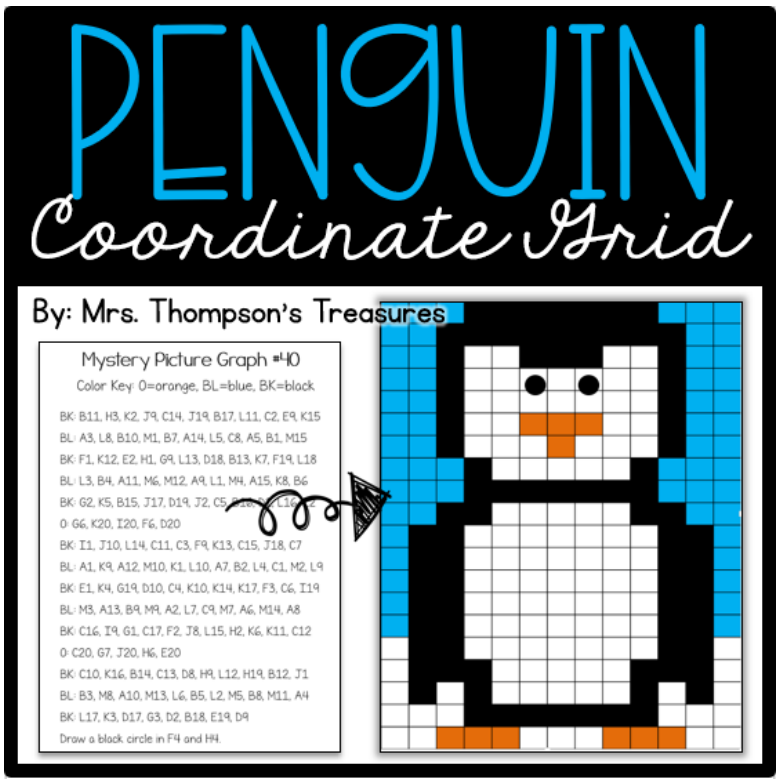
- A 'better' approach might be to consider transmission in spatial (as well as temporal) domain
- We could also introduce some randomness to allow for individual differences / chance occurrence in transmission & recovery
- What if people die?
- What if people get reinfected?

- Cellular automata are non-physical statistical models that are useful for spatial problems
- Instead of percolation or diffusion equations, these are rule-based models
- They treat evolution of each discrete cell as dependent on the properties of neighbouring cells
- Allows for very complex scenarios, based primarily on a probabilistic rather than deterministic approach

```
for j in eachindex(z[2:n, 2:n]) # Cartesian indexing
    # define rules for spread

    if z[j] == IFC && t[j] > r  # finds whether infection period has elapsed
       z[j] = REC
       status[j] = 3
    end

    if z[j] == IFC && t[j] <= r # calcs infection duration
       t[j] = t[j] + 1
       status[j] = 2
       if rand(D) == DR   #  predicts deaths as percentage of infected
           z[j] = REC*2
           status[j] = 4
       end
    end

    if z[j] >= thresh && z[j] <= IFC && t[j] <= r # calcs proximal transmission
       #t[j] = t[j] + 1

       if z[j-n-1] < IFC ; z[j-n-1] = rand(S) ; end  #min(5, z[j-n-1] + R0)
       if z[j-n] < IFC ; z[j-n] = rand(S) ; end #min(5, z[j-n] + R0)
       if z[j-n+1] < IFC ; z[j-n+1] = rand(S) ; end #min(5, z[j-n+1] + R0)
       if z[j-1] < IFC ; z[j-1] = rand(S) ; end #min(5, z[j-1] + R0)
       if z[j+1] < IFC ; z[j+1] = rand(S) ; end #min(5, z[j+1] + R0)
       if z[j+n-1] < IFC ; z[j+n-1] = rand(S) ; end #min(5, z[j+n-1] + R0)
       if z[j+n] < IFC ; z[j+n] = rand(S) ; end #min(5, z[j+n] + R0)
       if z[j+n+1] < IFC ; z[j+n+1] = rand(S) ; end #min(5, z[j+n+1] + R0)

    end


    if z[j] >= 1 && z[j] < IFC ; status[j] = 1 ; end
end
```

FROM FILE .....

TO FIGURE ...



4 | Outline

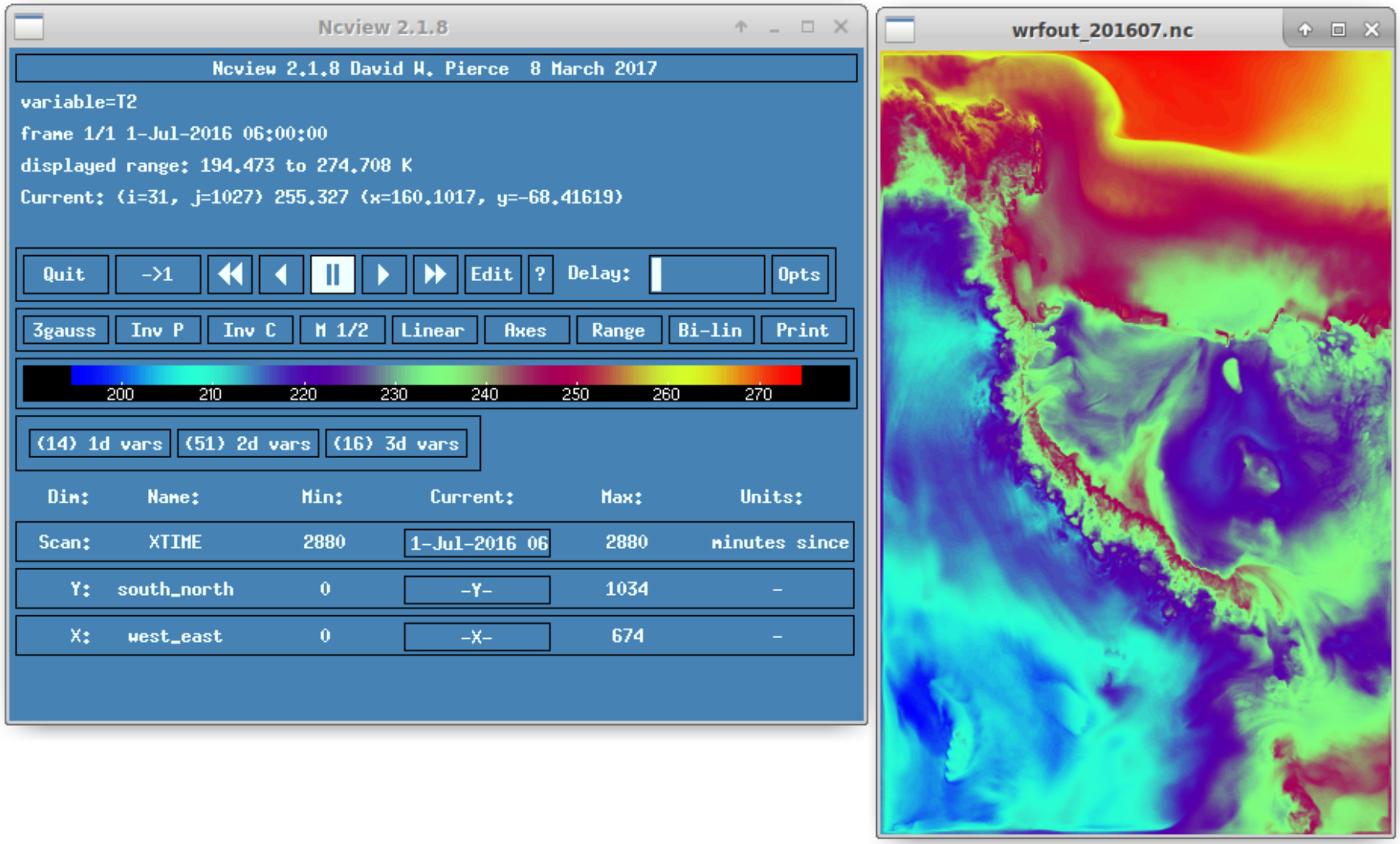
- Downloaded a file, so what?
- Play around with CDO
- Load into Python and visualise
- Make a publication figure

4 | Downloaded a file, so what?

4 | Downloaded a file, so what? - Using **ncview**

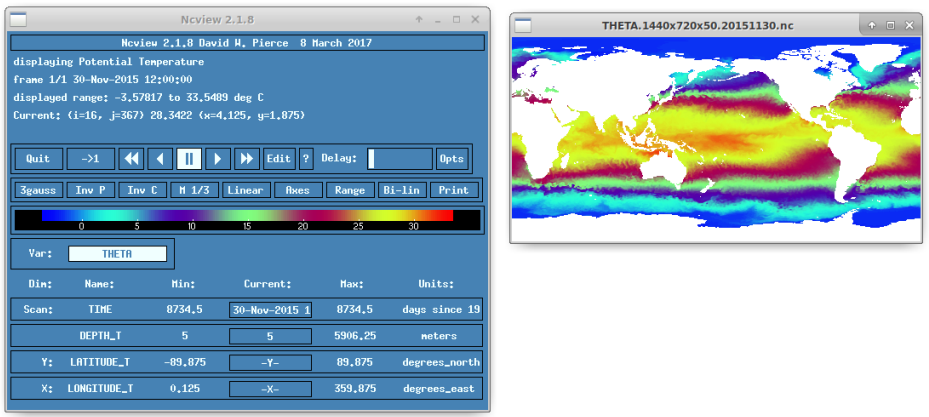
- What are the dimensions? what is the domain?
- What are the variables?
- What are the units? and range?
- Is there masked data? what is the value used?
- How to create a quick timeseries?

4 | Downloaded a file, so what? - Using **ncview**

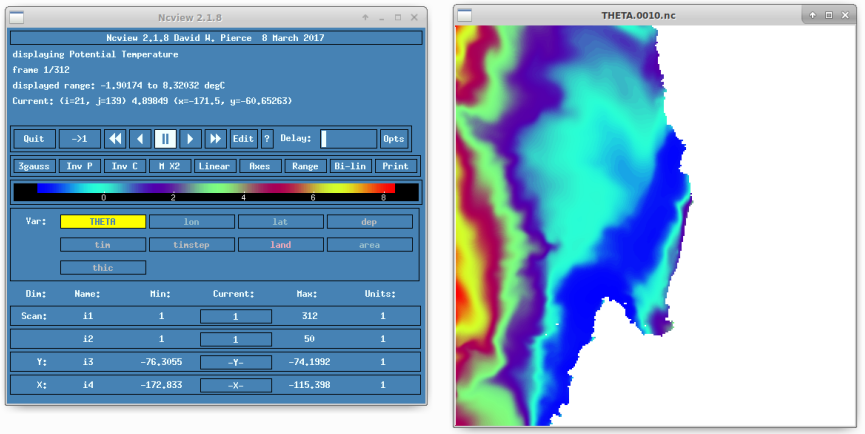


WRF output, 2m temperature

4 | Downloaded a file, so what? - Using **ncview**

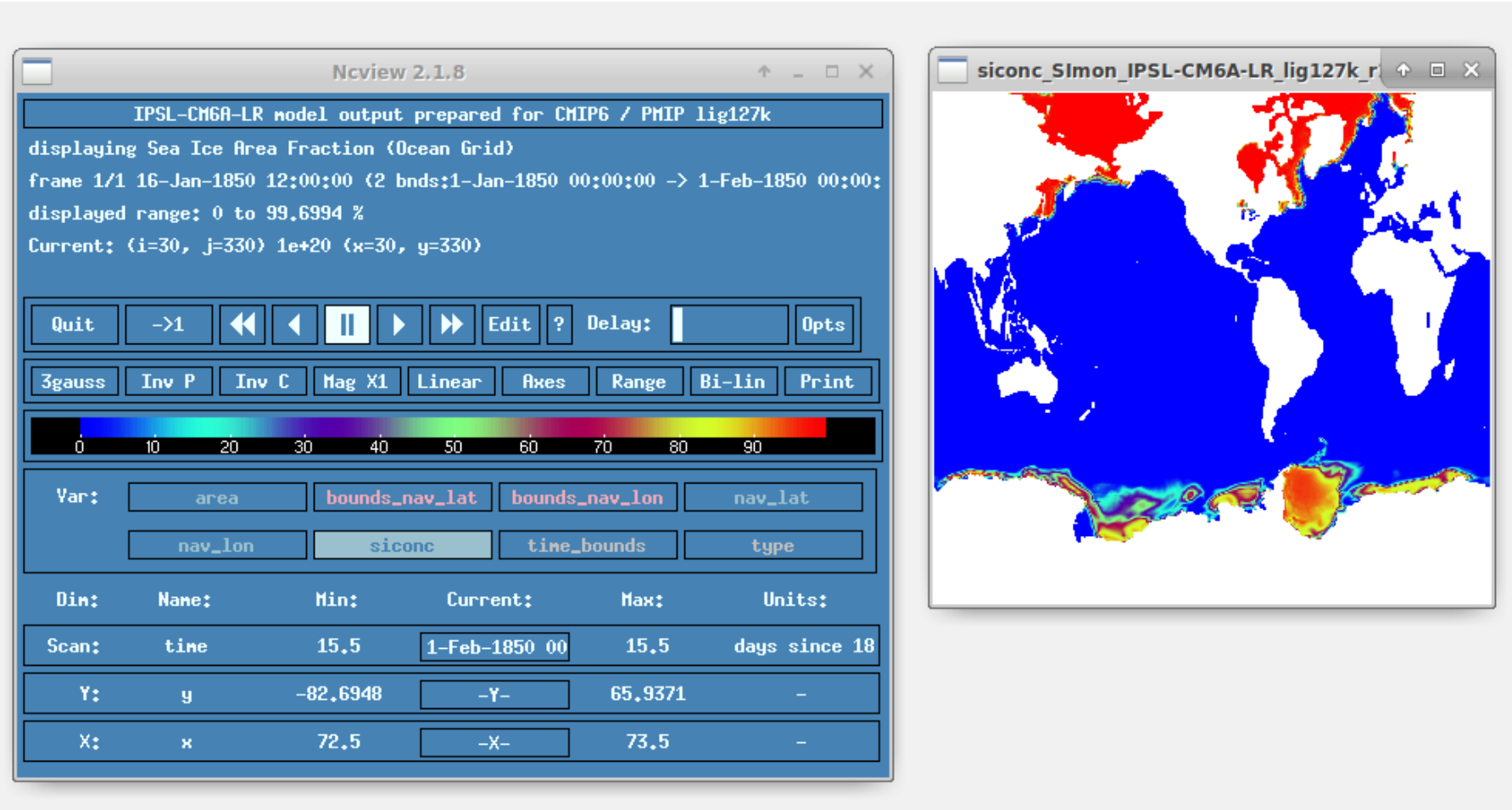


ECCO2, potential temperature



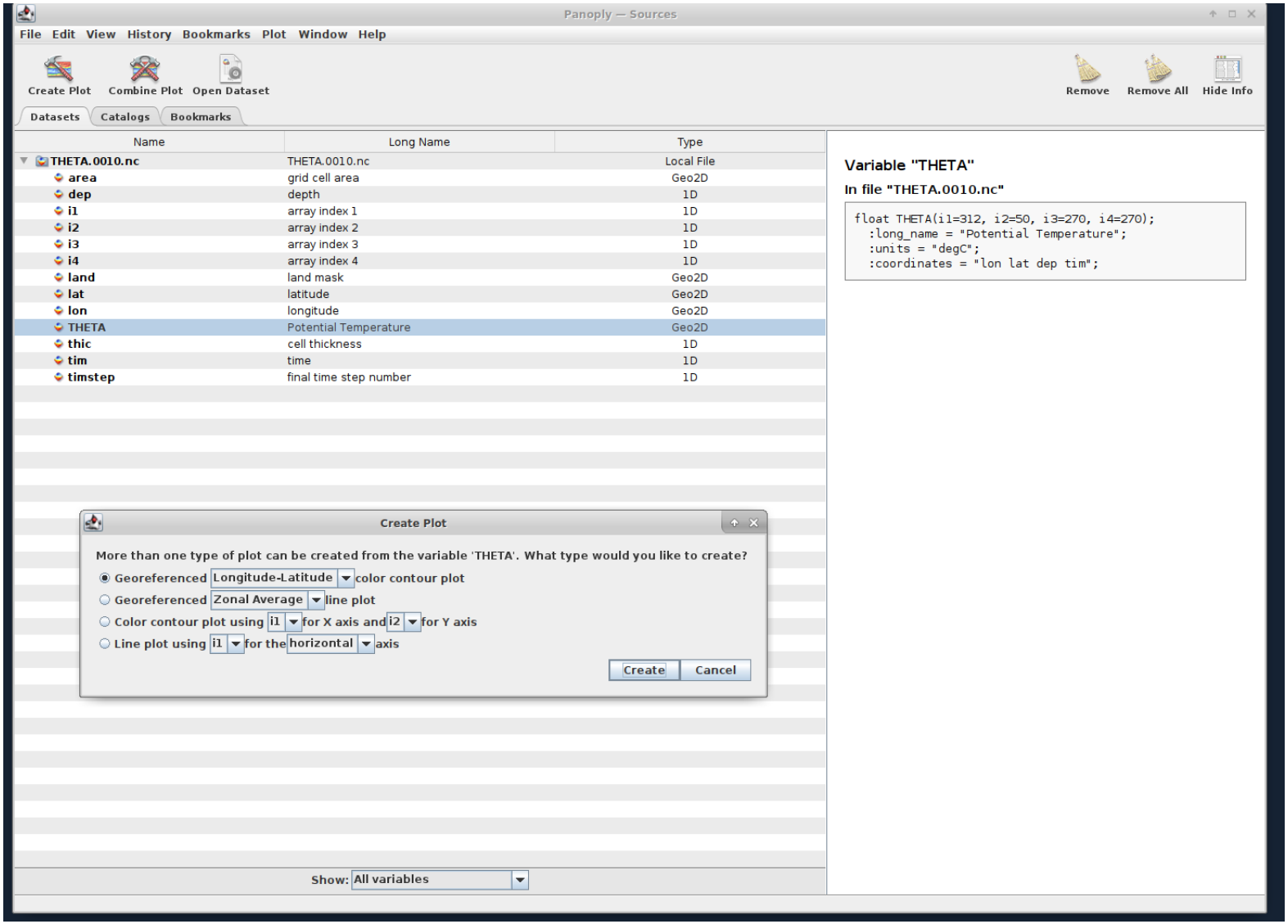
ECCO v5, potential temperature

4 | Downloaded a file, so what? - Using **ncview**

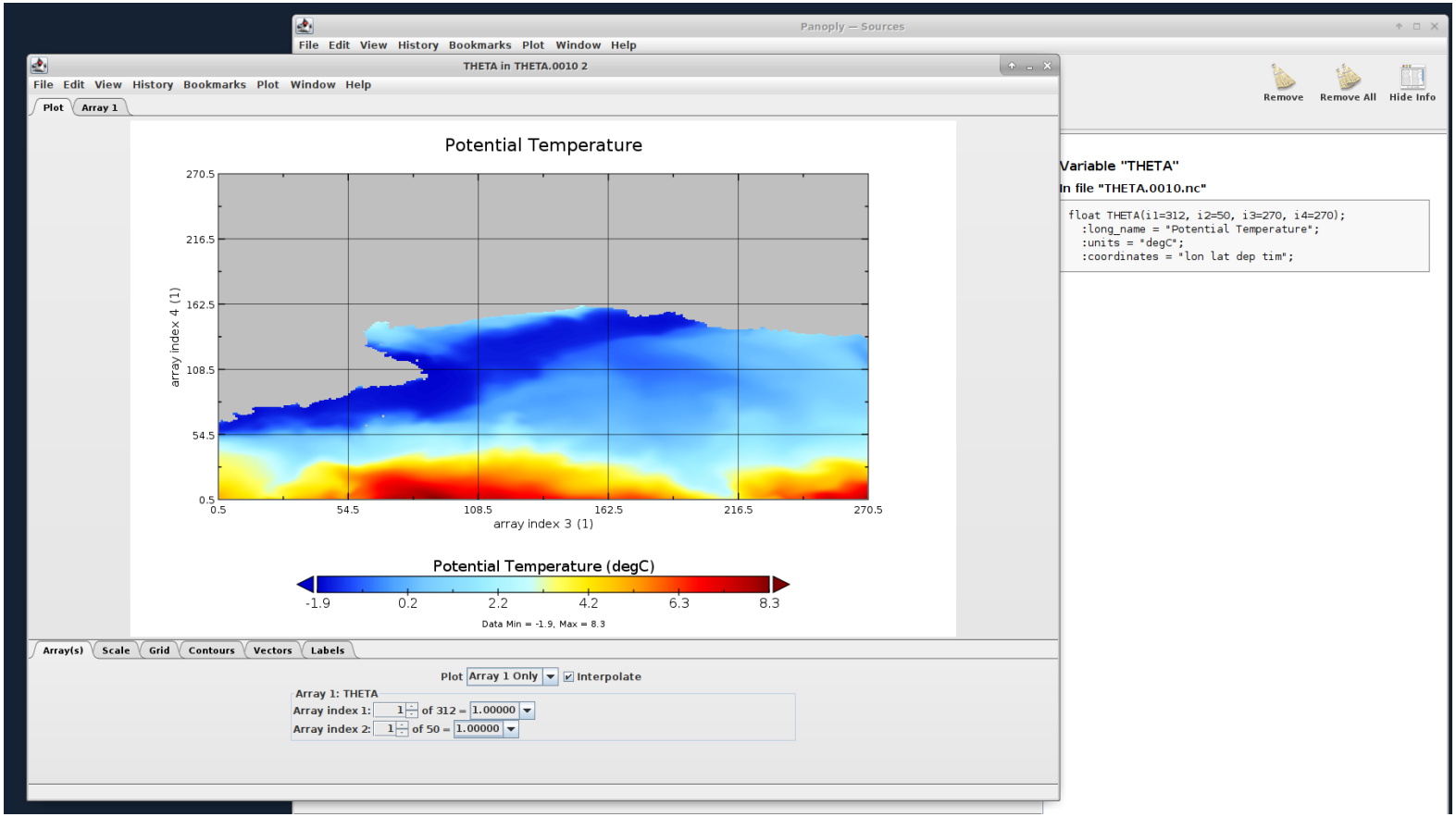


IPSL-CM6A-LR, sea ice area fraction

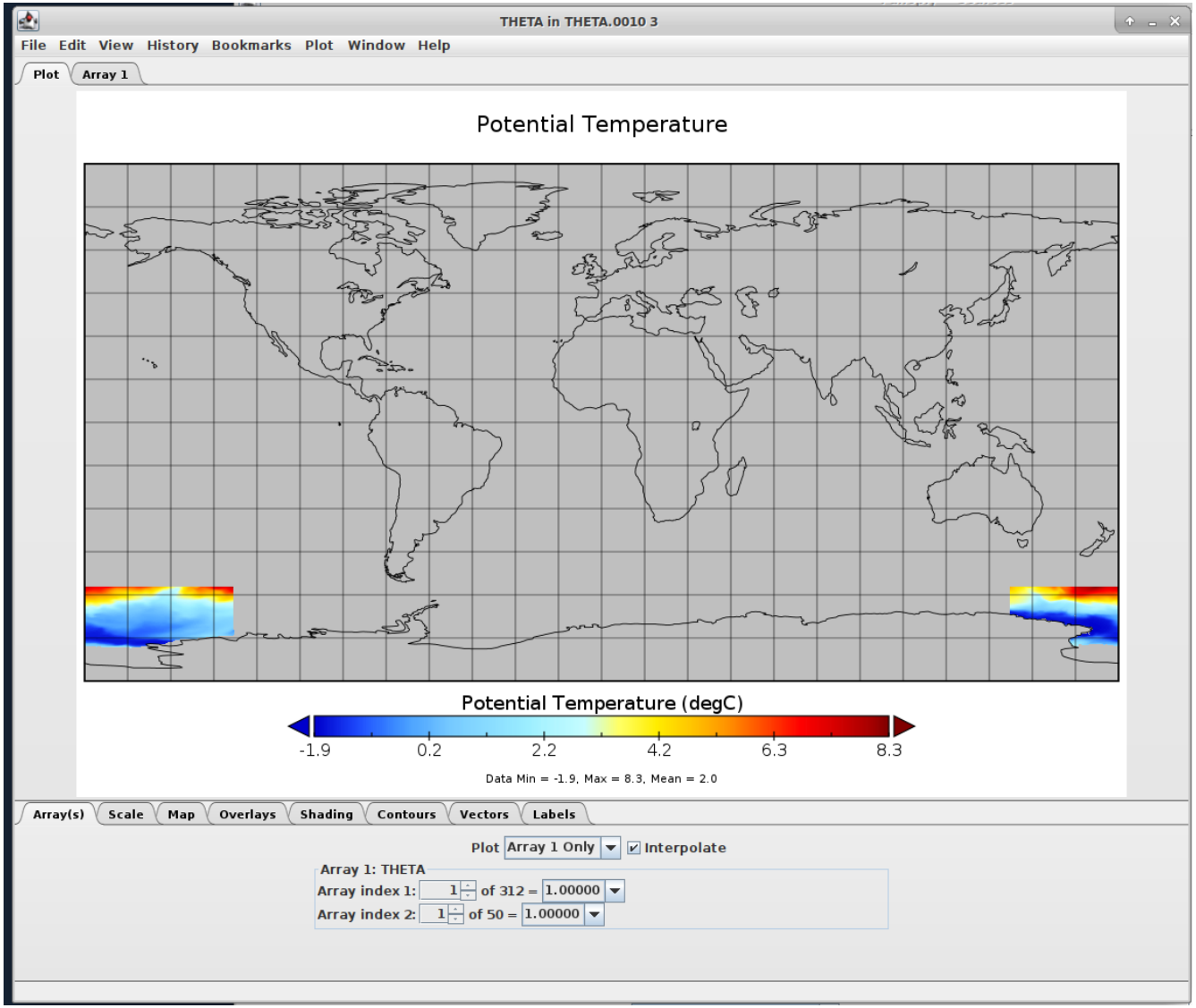
4 | Downloaded a file, so what? - Using **panoply**



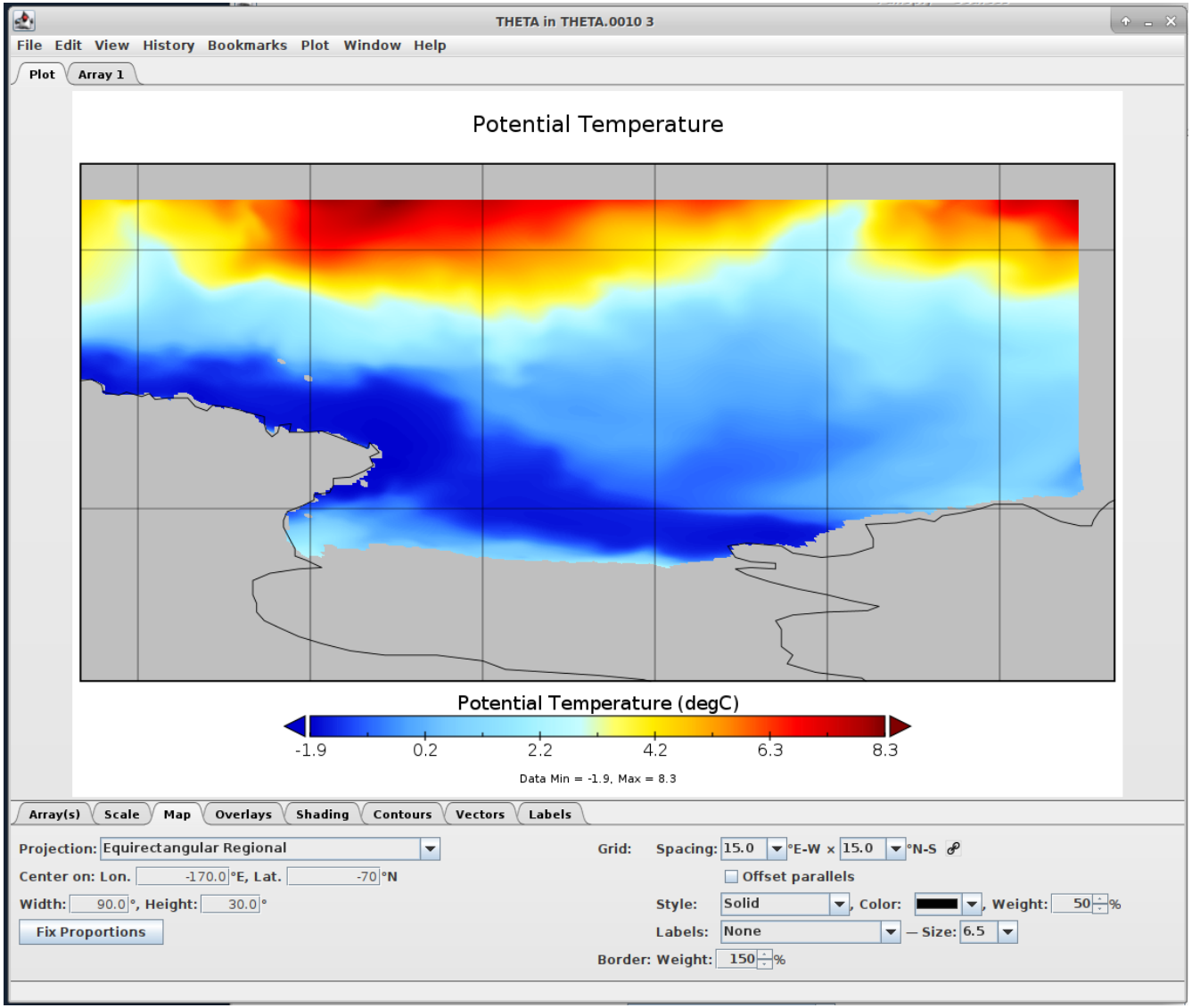
4 | Downloaded a file, so what? - Using **panoply**



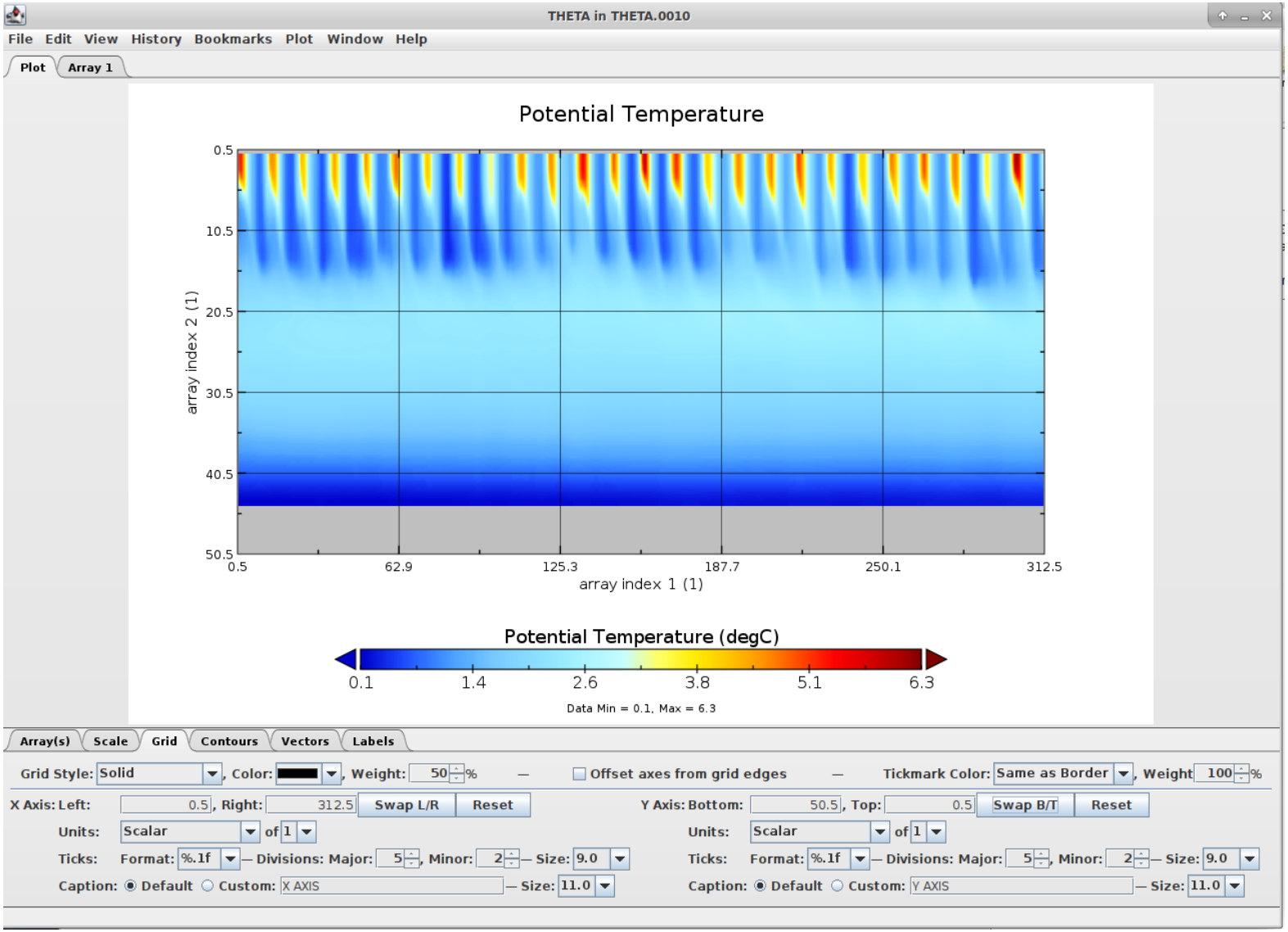
4 | Downloaded a file, so what? - Using **panoply**



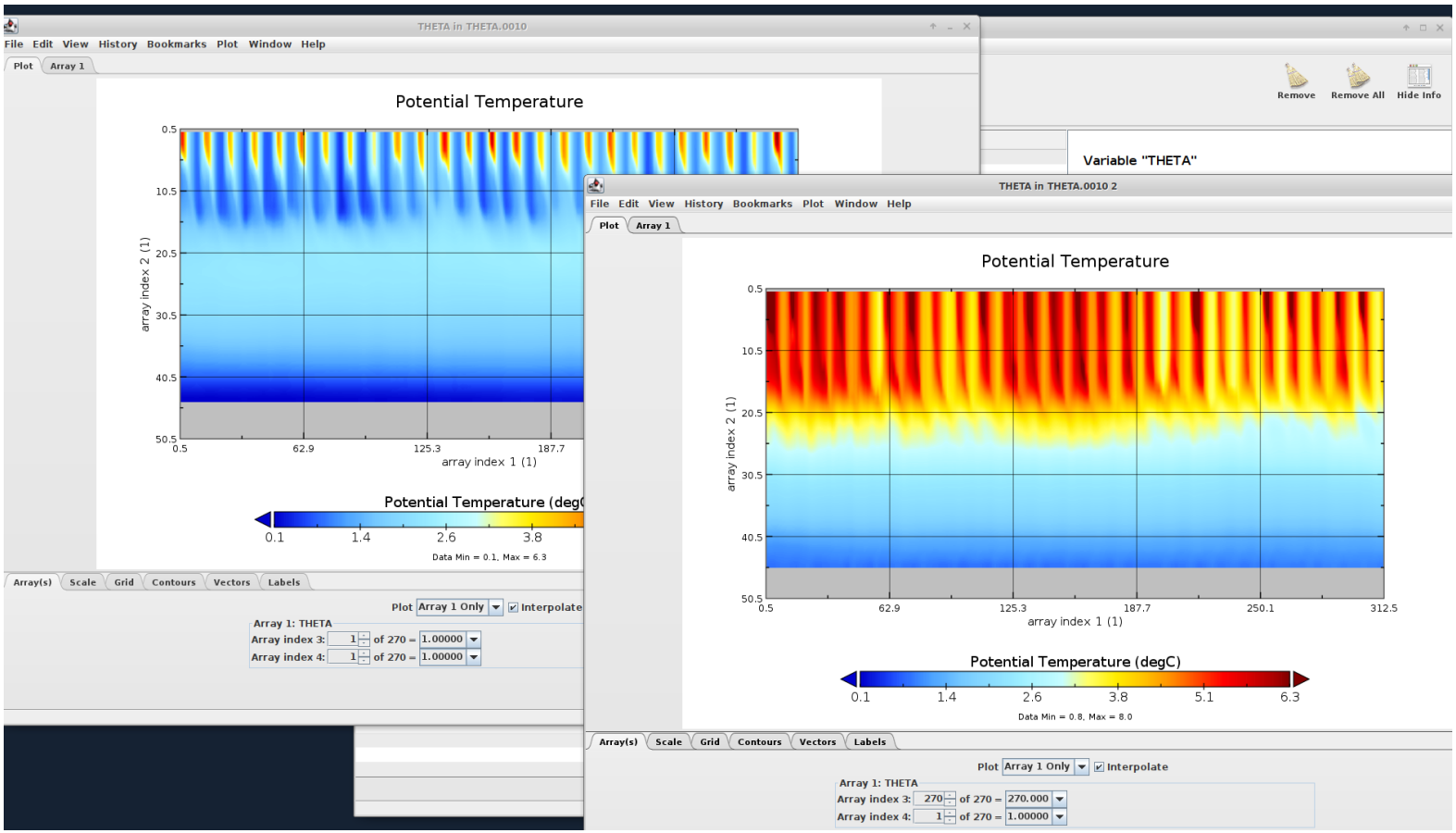
4 | Downloaded a file, so what? - Using **panoply**



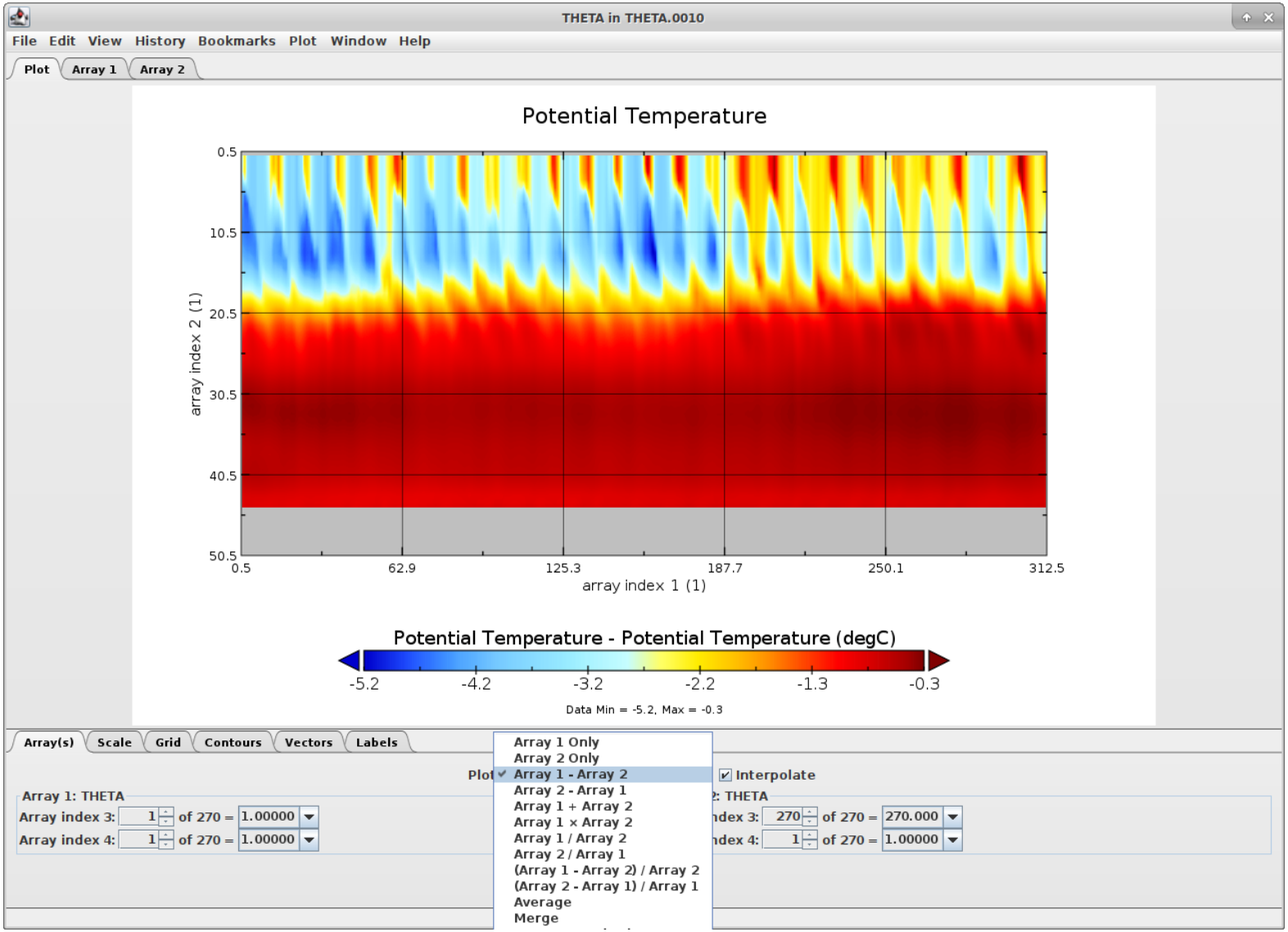
4 | Downloaded a file, so what? - Using **panoply**



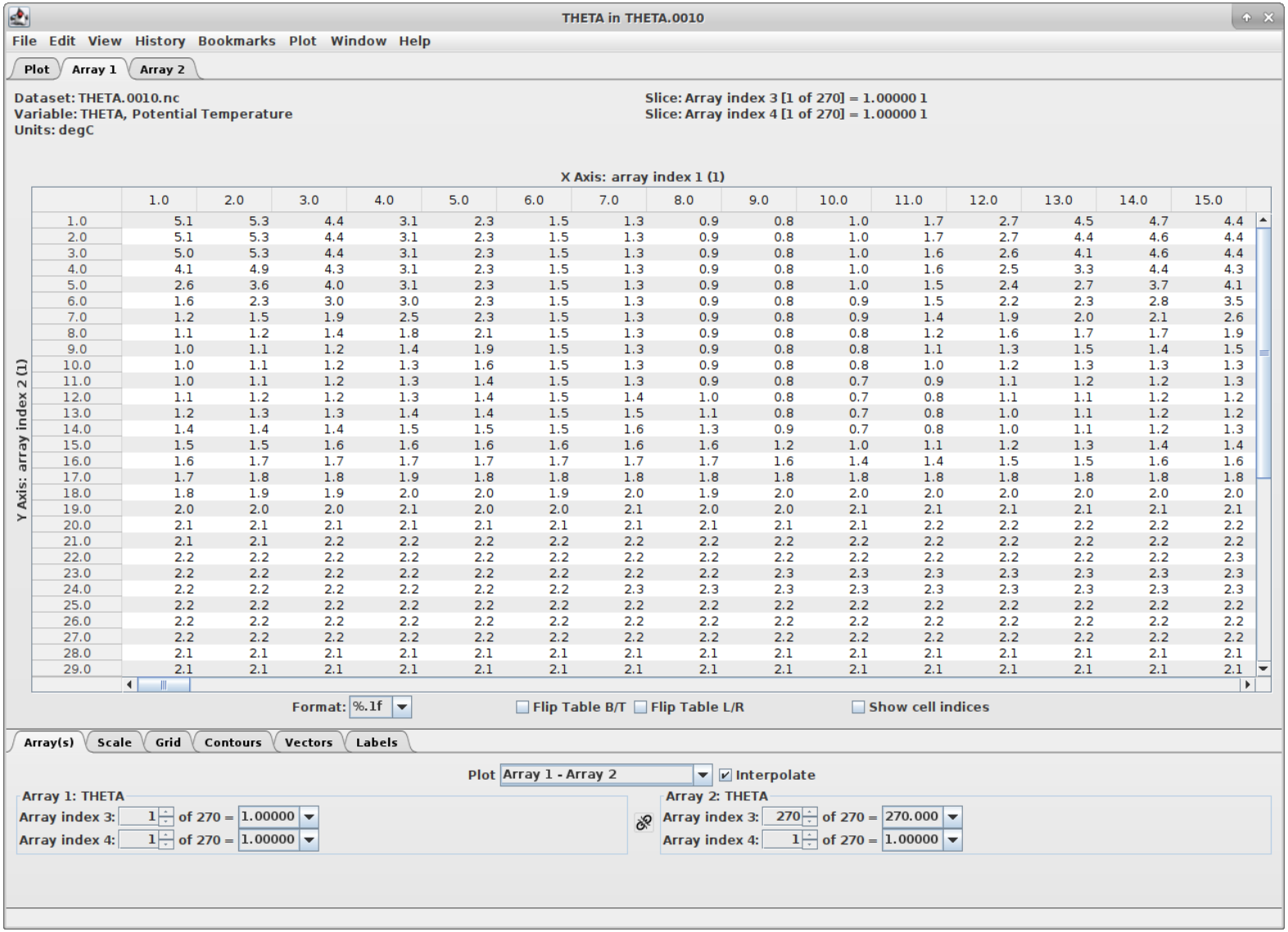
4 | Downloaded a file, so what? - Using **panoply**



4 | Downloaded a file, so what? - Using **panoply**



4 | Downloaded a file, so what? - Using **panoply**

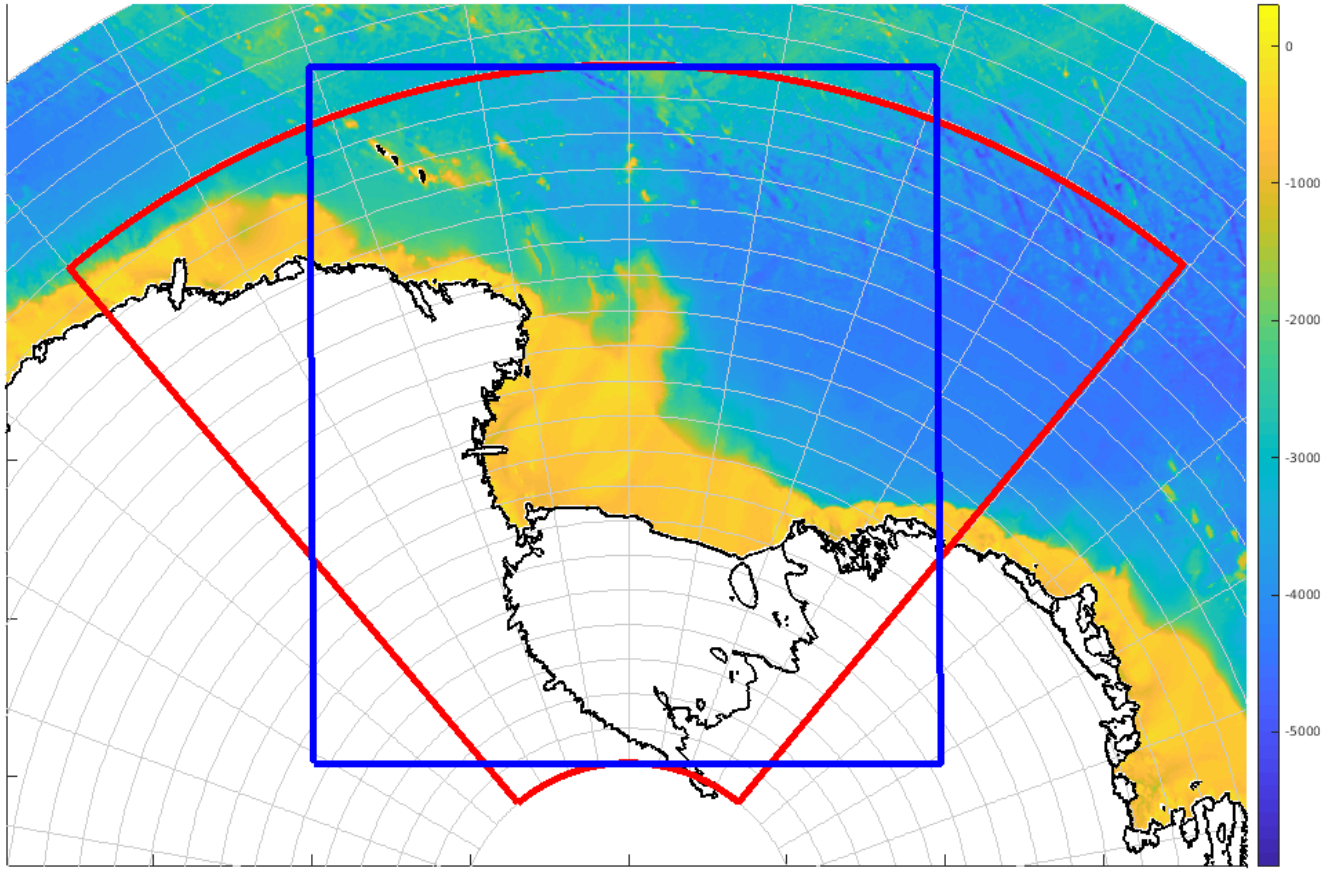


4 | Downloaded a file, so what? - **coordinates**

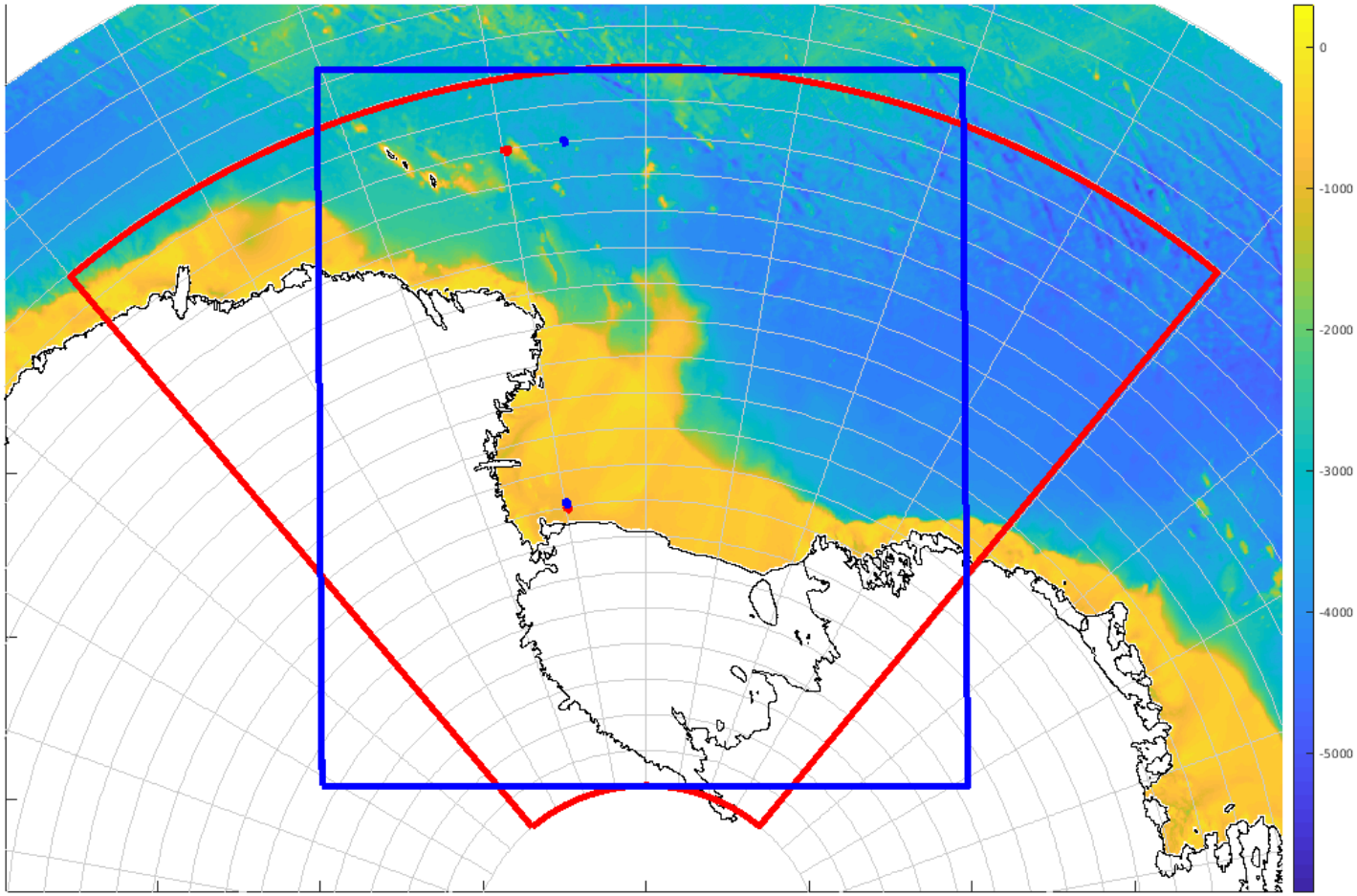
Different coordinate systems:

- lon-lat (easy to figure out where we are)
- x-y (equidistant grid in km, better at the poles)
- irregular grid (removes the pole problem)

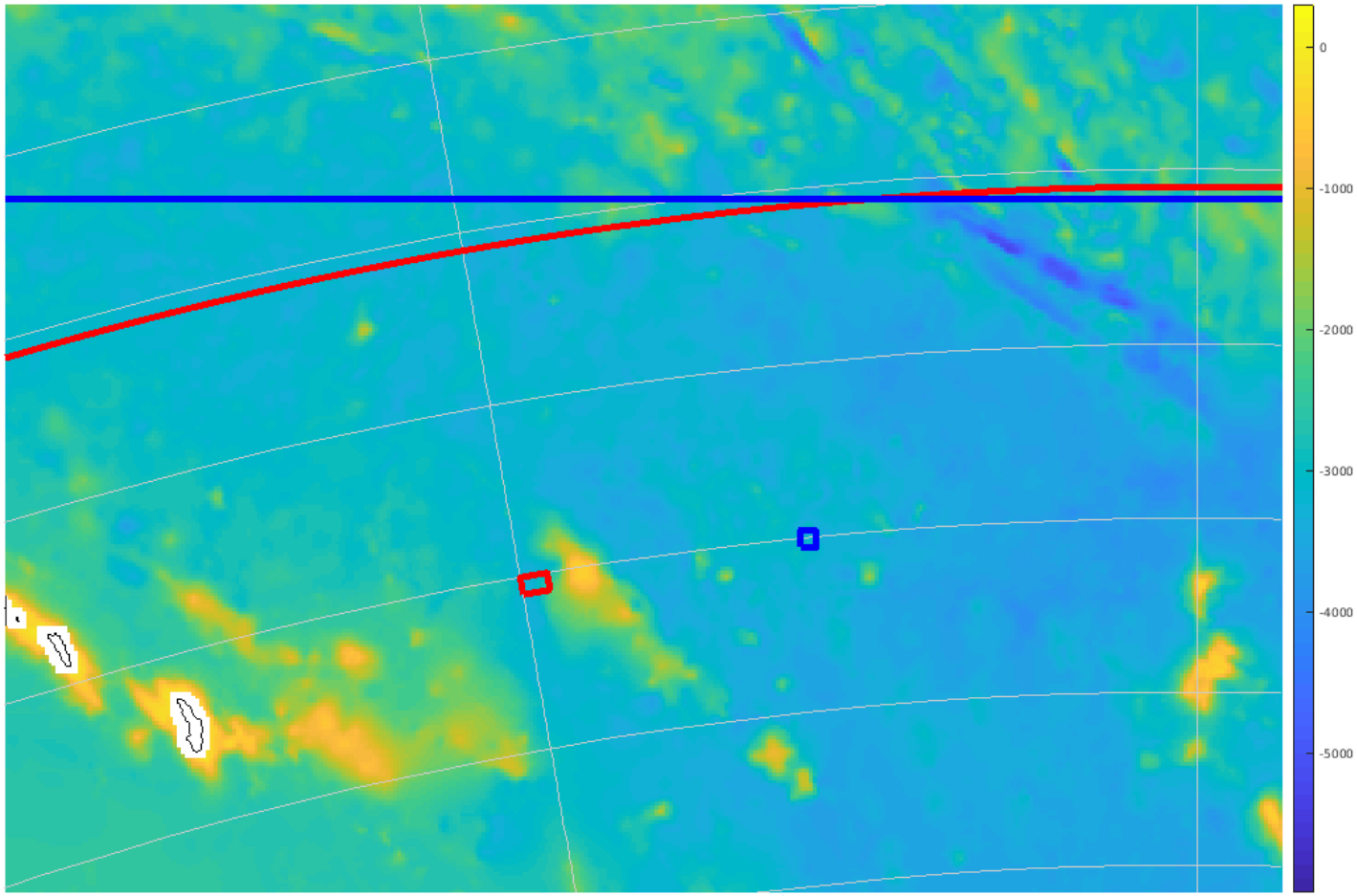
4 | Downloaded a file, so what? - **coordinates**



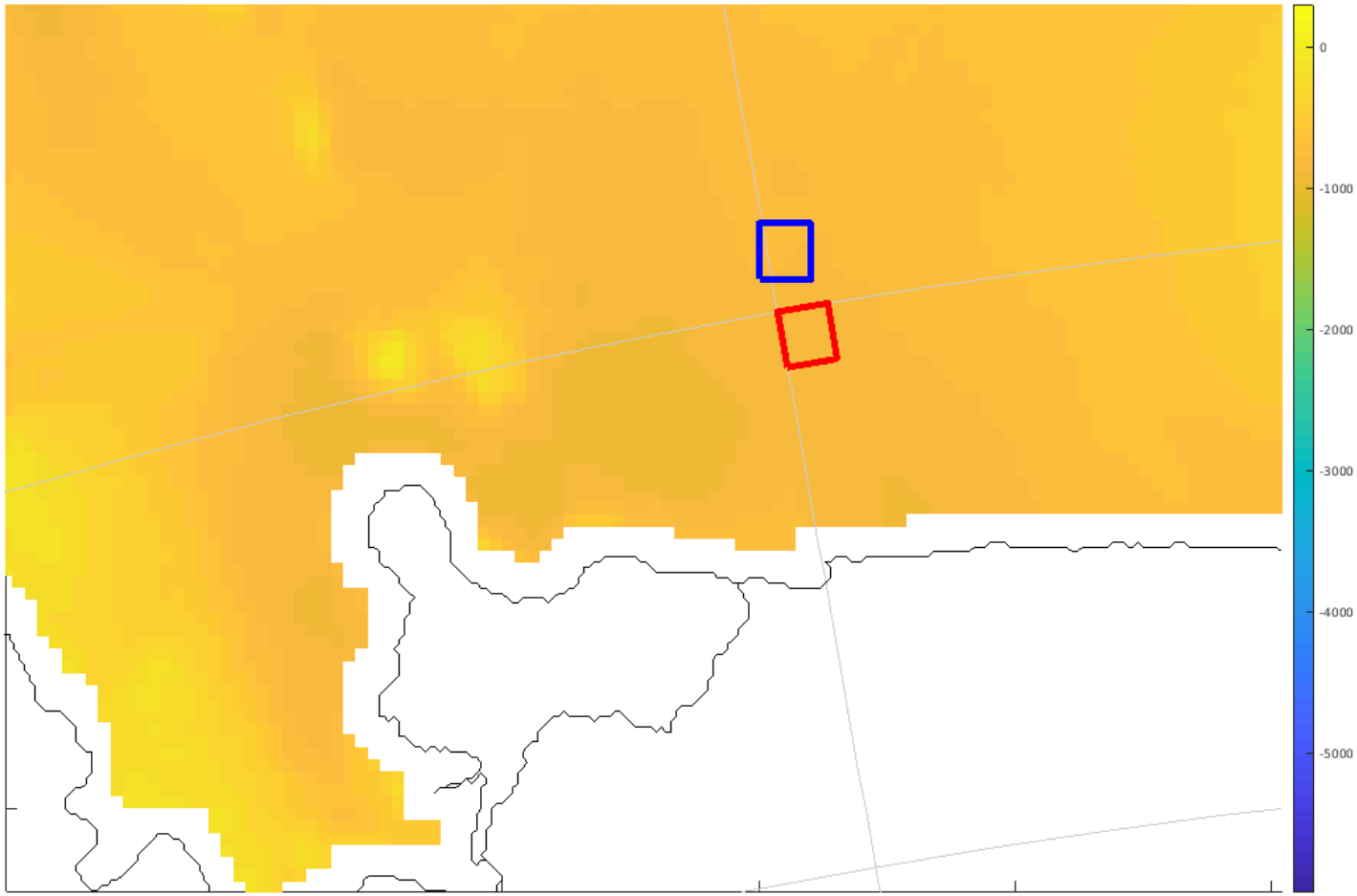
4 | Downloaded a file, so what? - **coordinates**



4 | Downloaded a file, so what? - **coordinates**



4 | Downloaded a file, so what? - **coordinates**



4 | Downloaded a file, so what? - get information on the data

- The naming **conventions** are useful: what do the two files contain?

- Let's verify using **ncview** or **cdo**:
  - *ncdump filename*
  - *ncdump -h filename*
  - *cdo sinfo filename*

Pop quiz: open the theta....2013.nc file and:

- what is the variable(s)? short and long name?
- what is the spatial resolution of the data? and the dimension of the file?
- what is the temporal coverage of the data?
- is there a mask value anywhere?
- how many dimensions does this variable have?

4 | Play around with **cdo**

4 | Play around with **cdo**

Manipulating big files can be slow or make your program crash!

Pre-processing can save our life: let's use 

cdo can be used to :

- select
  - spatially
  - temporally
  - vertically
- mask
- remap
- do statistics and calculations
- and so much more...

4 | Play around with **cdo** - spatial extent

To reduce the **spatial** extent:

- *cdo sellonlatbox,lon0,lon1,lat0,lat1 infile outfile*
- *cdo selindexbox,index0,index1,index0,index1 infile outfile*

example: to select New Zealand (to be done on both theta and so files)

cdo sellonlatbox,150,180,-30,-50
thetao_Omon_CESM2_historical_r1i1p1f1_gr_201301-201412.nc NZ_theta.nc

4 | Play around with **cdo** - temporal extent

To reduce the **temporal** extent:

- *cdo selmon,1 infile outfile*
- *cdo select,timestep=1 infile outfile*
- *cdo splityear infile prefix-*

4 | Play around with **cdo** - vertical (level) extent

To reduce the number of **levels**:

- *cdo sellevel,1 infile outfile*
- *cdo select,levrange=lev1,lev2,name=varname*

example: let's select the second and third levels

cdo select,level=10,20,name=thetao NZ_theta.nc NZ_theta_levels.nc

and let's do the same for salinity.

4 | Play around with **cdo** - stats, remap and mask

But also to do stats, remap and mask data...

- do some **stats**:
  - *cdo monmean ...*
  - *cdo ymonmean ...*
  - *cdo add(c) ..., cdo sub(c) ...*
  - *cdo monstd ...*
- *cdo remapbil ...* will use bilinear interpolation to **remap** one grid ont another (! create a grid description file using *cdo griddes* beforehand)
- *cdo ifthenelse ...* will use a condition to determine if the **mask** is used (! create a mask file beforehand)

  example: Let's mask out the areas with salinity > 34.5 for the temperature file:

  cdo mulc,0 NZ_theta_levels.nc zeroes.nc
  cdo ifthenelse -gec,34.5 NZ_tso_levels.nc zeroes.nc NZ_theta_levels.nc test.nc

4 | Play around with **cdo** - pop quiz

Pop quiz: Use **cdo** commands to compare

- two cross sections
- of mean June
- sea surface temperature
- over the dateline
- using the two thetao files (201301-201412 and 185001-185112)
- using the red-blue colormap
- with the surface at the top
- and a colorbar symmetrical around 0

Hint: subtract one from the other

4 | Load into Python and visualise

4 | Load into Python and visualise - essentials

Now, time to create a script to make reproducible plots:

- let's create a python script (*decide_on_a_relevant_and_clear_name.**py***)
- one figure per file is a useful rule
- add comments to explain what we do
- prepare our data before loading it
- to run a python script : *python scriptname.py* in the terminal

4 | Load into Python and visualise - packages

Let's import the relevant python packages:

```
# -*- coding: utf-8 -*-
#
# Winterschool 2021 - Alena and Alex
#
# Analysis and plotting of spatial data
##################################################

import numpy as np  # This will import the python numerical
# library. It can be utilised to perform a number of
# mathematical operations on arrays such as trigonometric,
# statistical, and algebraic routines.

from netCDF4 import Dataset # This will enable to read in
# netcdf datasets.

import matplotlib.pyplot as plt # This is an interface to
# provide a MATLAB-like way of plotting

from batlow import *    # to have a *fancy* colormap
```

4 | Load into Python and visualise - read a netCDF

Let's import the netcdf dataset and get a sense of the dimensions:

```
# read in the data
filename='thetao_Omon_CESM2_historical_r1i1p1f1_gr_201301-201412.nc'
ncfile = Dataset(filename)

var = ncfile.variables['thetao'][0,0,:,:]

print(np.shape(var))
```

(24, 33, 180, 360)

Pop quiz:
To which dimensions do these numbers correspond?

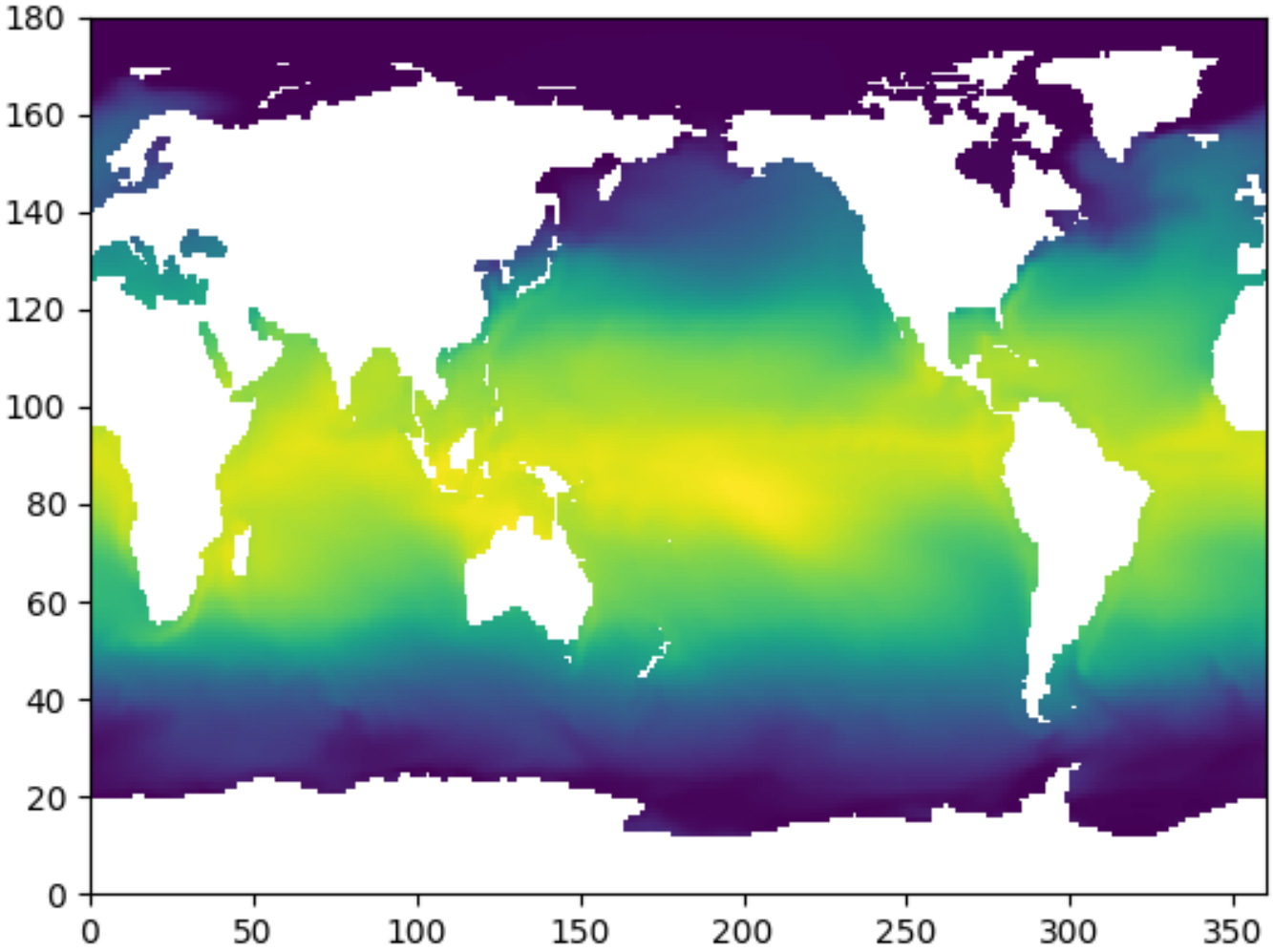
4 | Load into Python and visualise - read a netCDF

Let's get the dimensions we want and plot the data:

we want to look at the surface, first time step and whole spatial extent of the file

```
# read in the data
filename='thetao_Omon_CESM2_historical_r1i1p1f1_gr_201301-201412.nc'
ncfile = Dataset(filename)

var = ncfile.variables['thetao'][0,0,:,:]

# plot

plt.pcolor(var)

# make plot appear on screen

plt.show()
```

4 | Load into Python and visualise - simple plot

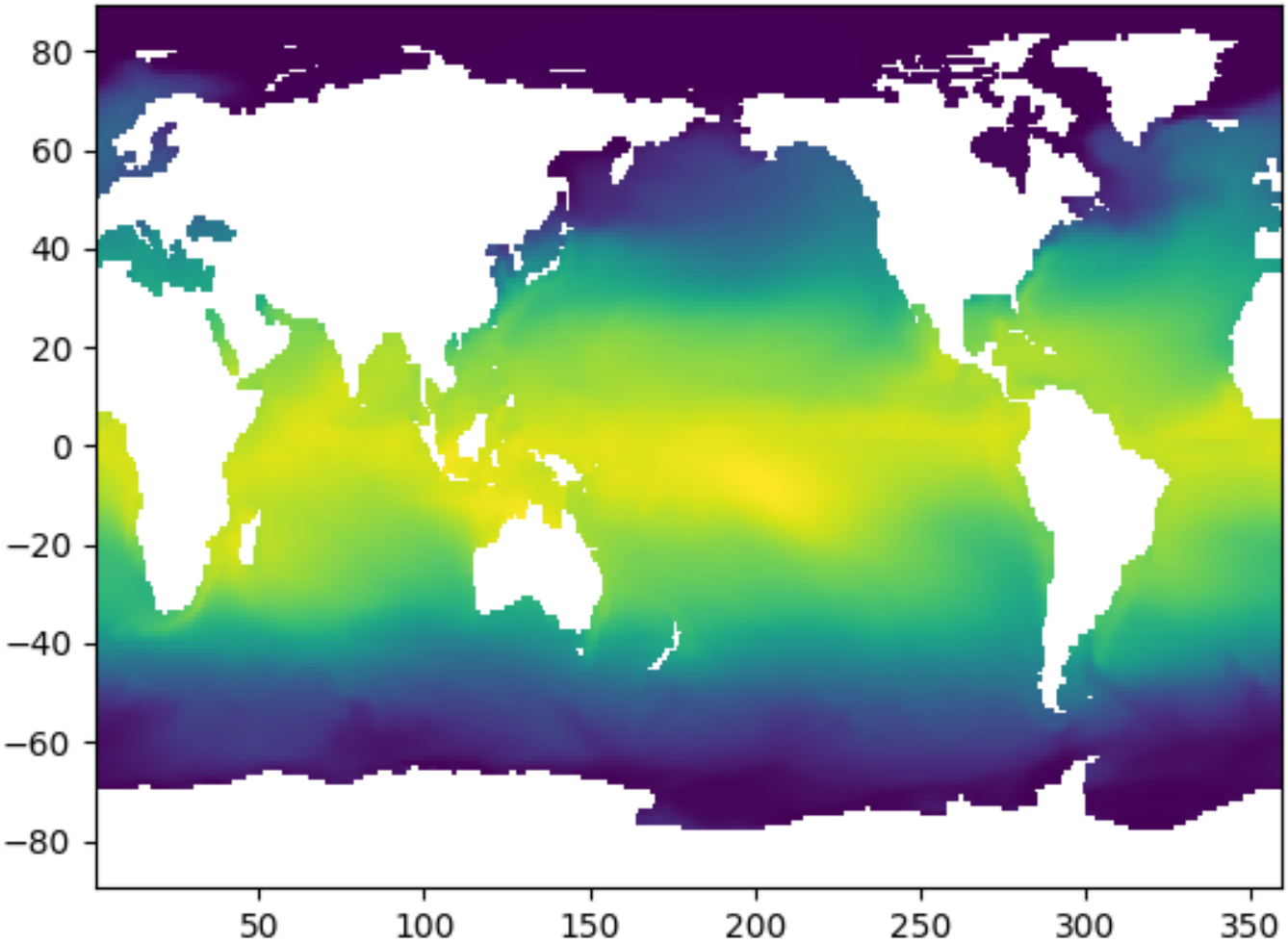


4 | Load into Python and visualise - simple plot

Let's add the latitude and longitude coordinates:

```
lat = ncfile.variables['lat']
lon = ncfile.variables['lon']


# add the X and Y dimensions to plot

plt.pcolor(lon,lat,var)
```

4 | Load into Python and visualise - simple plot



4 | Load into Python and visualise - colormap

Perspective | Open Access | Published: 28 October 2020

The misuse of colour in science communication

Fabio Crameri, Grace E. Shephard & Philip J. Heron

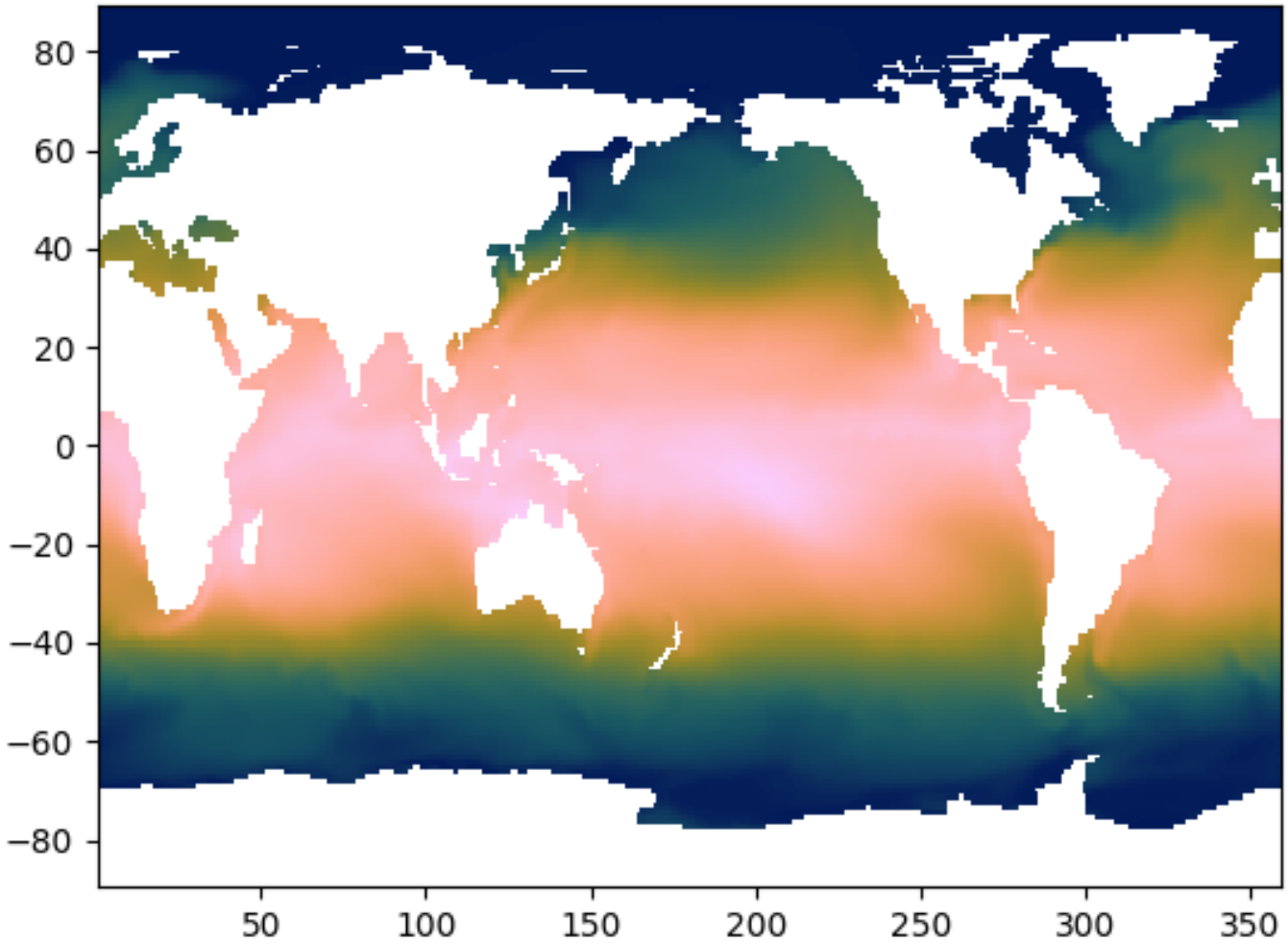
Nature Communications **11**, Article number: 5444 (2020) | Cite this article

Let's add a *fancy* colormap:

```
from batlow import *    # to have a *fancy* colormap

plt.pcolor(lon,lat,var,cmap=batlow_map)

plt.show()
```

(!! we need to add the batlow.py to the working directory)

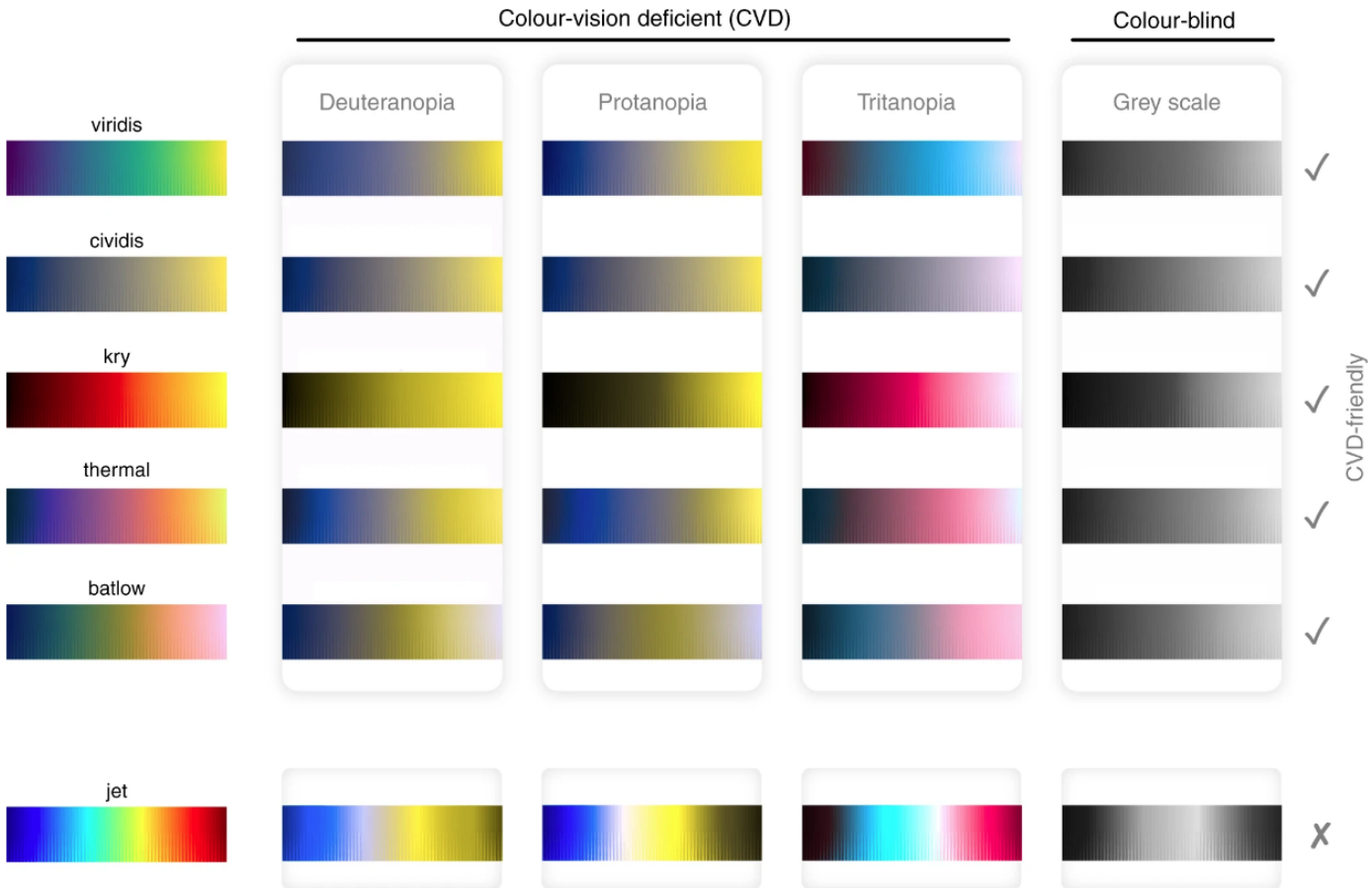
4 | Load into Python and visualise - colormap



4 | Load into Python and visualise - colormap

Fig. 2: Colour vision tests.

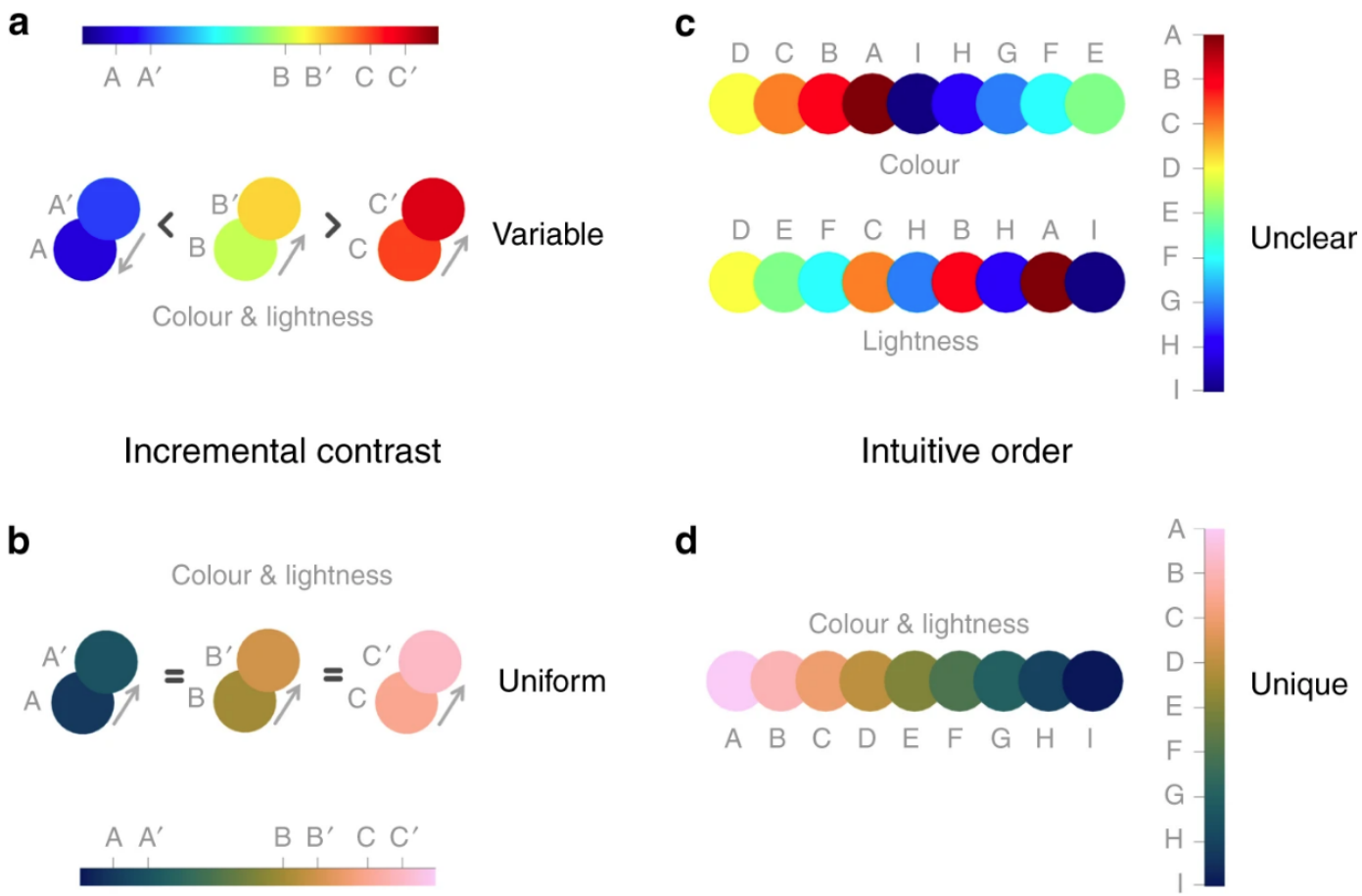
From: The misuse of colour in science communication



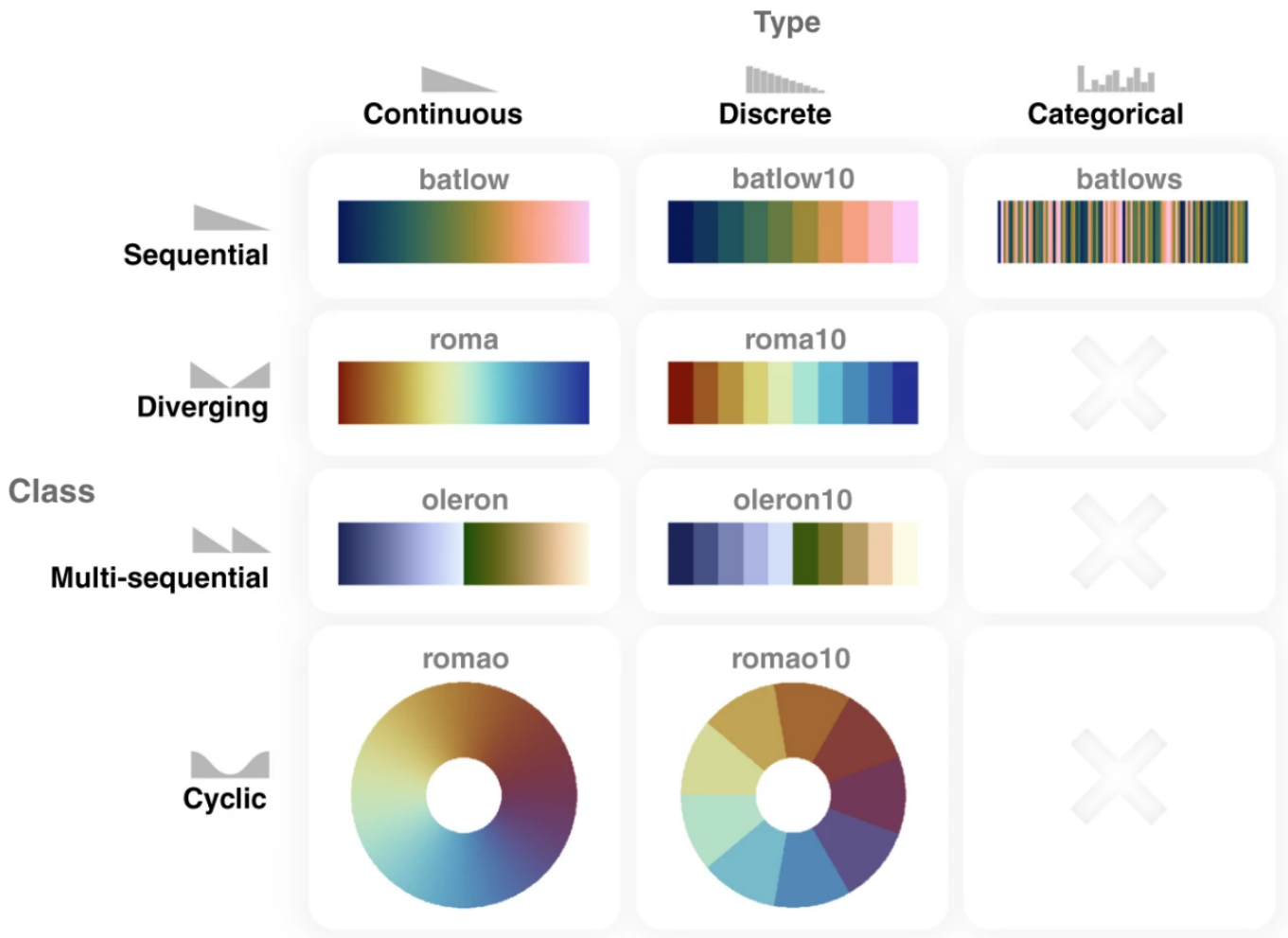
4 | Load into Python and visualise - colormap

Fig. 4: Perceptual uniformity and order.

From: The misuse of colour in science communication



3 | Load into Python and visualise - colormap

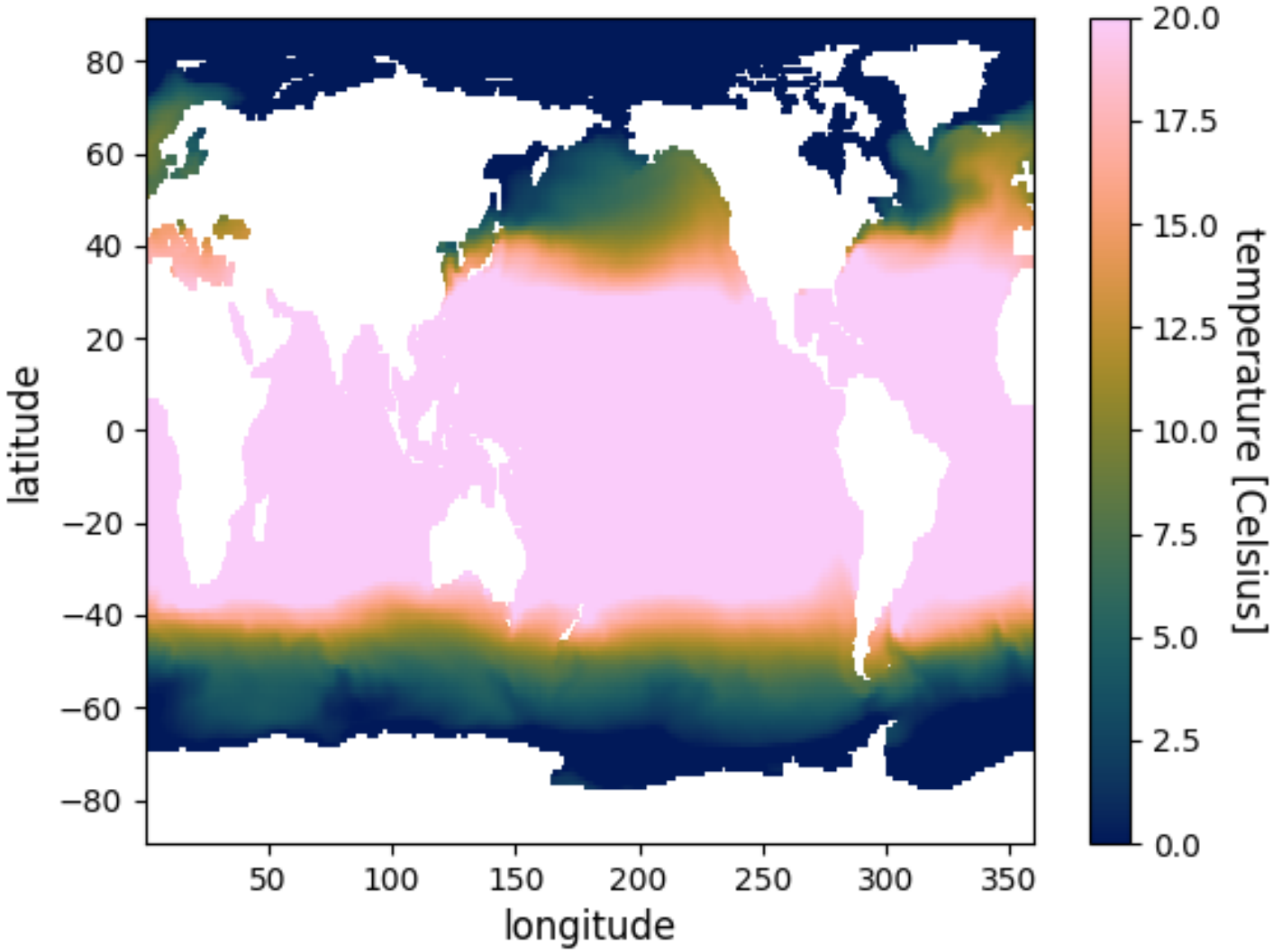


4 | Load into Python and visualise - labels and colorbar

Let's add labels and a colorbar limited at 20°C, but pay attention to the variable names and units!

```
ftsize = 12

# add axes labels
plt.xlabel('longitude',fontsize = ftsize)
plt.ylabel('latitude',fontsize = ftsize)

# colorbar
cbar = plt.colorbar()
cbar.ax.get_yaxis().labelpad = 15
cbar.ax.set_ylabel('temperature [Celsius]',fontsize = ftsize,rotation=270)
plt.clim(0,20)
```

4 | Load into Python and visualise - labels and colorbar



4 | Load into Python and visualise - masking

And, if we wanted to mask out some data, we can use **cdo** as done previously
But it is also possible in python:

To mask out a certain value:

```
## make a mask

var= np.ma.MaskedArray(var,mask=[(a>10) for a in var])
```

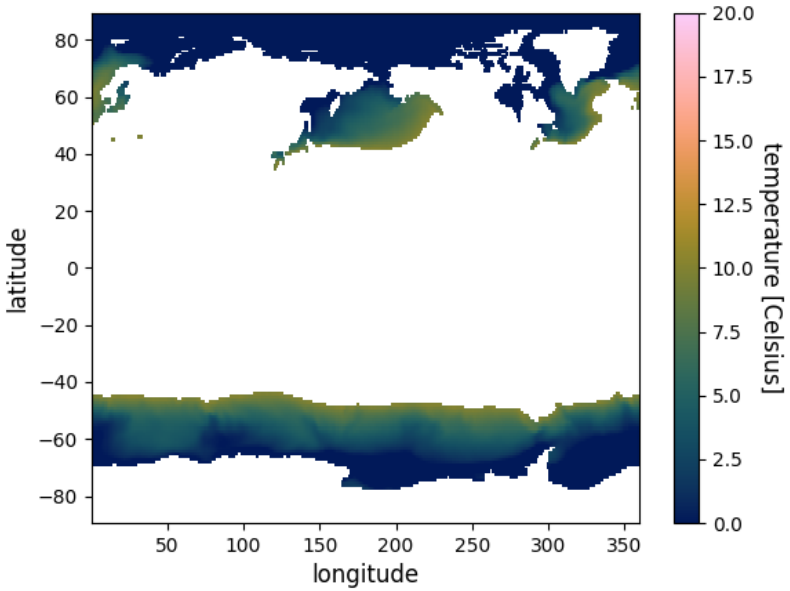
and to mask an area based on lon/lat: lon and lat are 1D variables, we need to make them 2D

```
## make a mask

#var= np.ma.MaskedArray(var,mask=[(a>10) for a in var])

# create a mesh grid
Lon,Lat=np.meshgrid(lon,lat)

# make a mask
var= np.ma.MaskedArray(var,mask=[(a>10) for a in Lat])
```

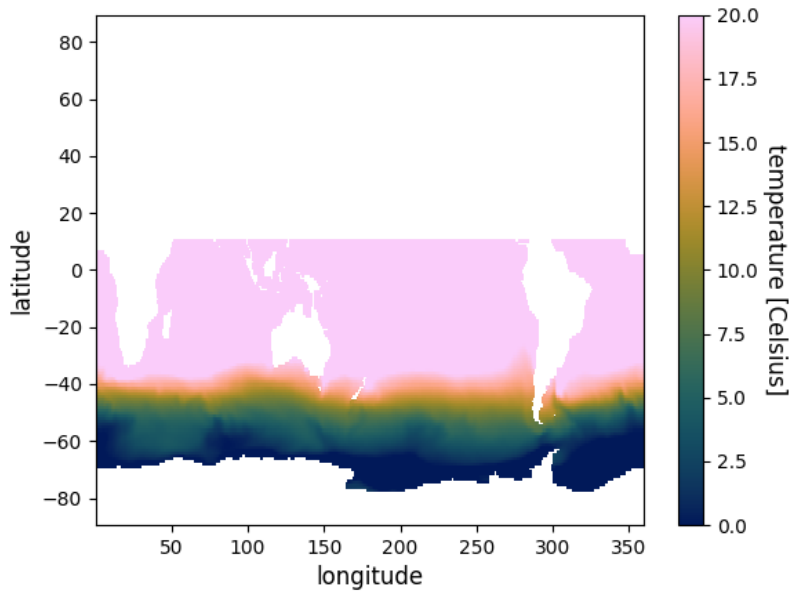
first:

4 | Load into Python and visualise - masking

To mask out a certain value:



and to mask an area based on lon/lat:

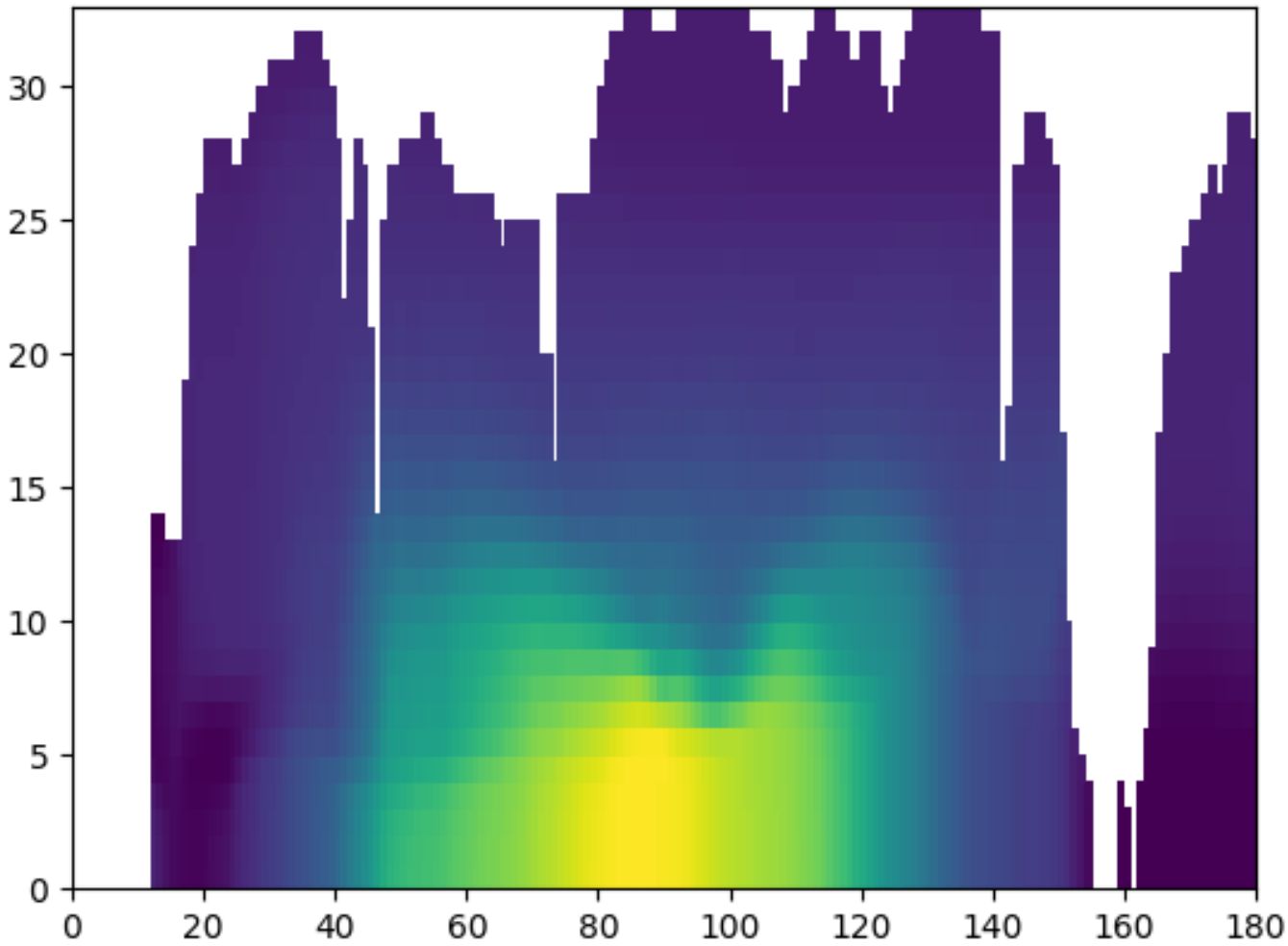


4 | Load into Python and visualise - cross-sections

We swap dimensions and plot along two dimensions (lon/time, depth/time,...).

```
# read in the data
filename='thetao_Omon_CESM2_historical_r1i1p1f1_gr_201301-201412.nc'
ncfile = Dataset(filename)

var = ncfile.variables['thetao'][0,:,:,180]

# plot
plt.pcolor(var)

# make plot appear on screen
plt.show()
```

4 | Load into Python and visualise - cross-sections



4 | Load into Python and visualise - cross-sections

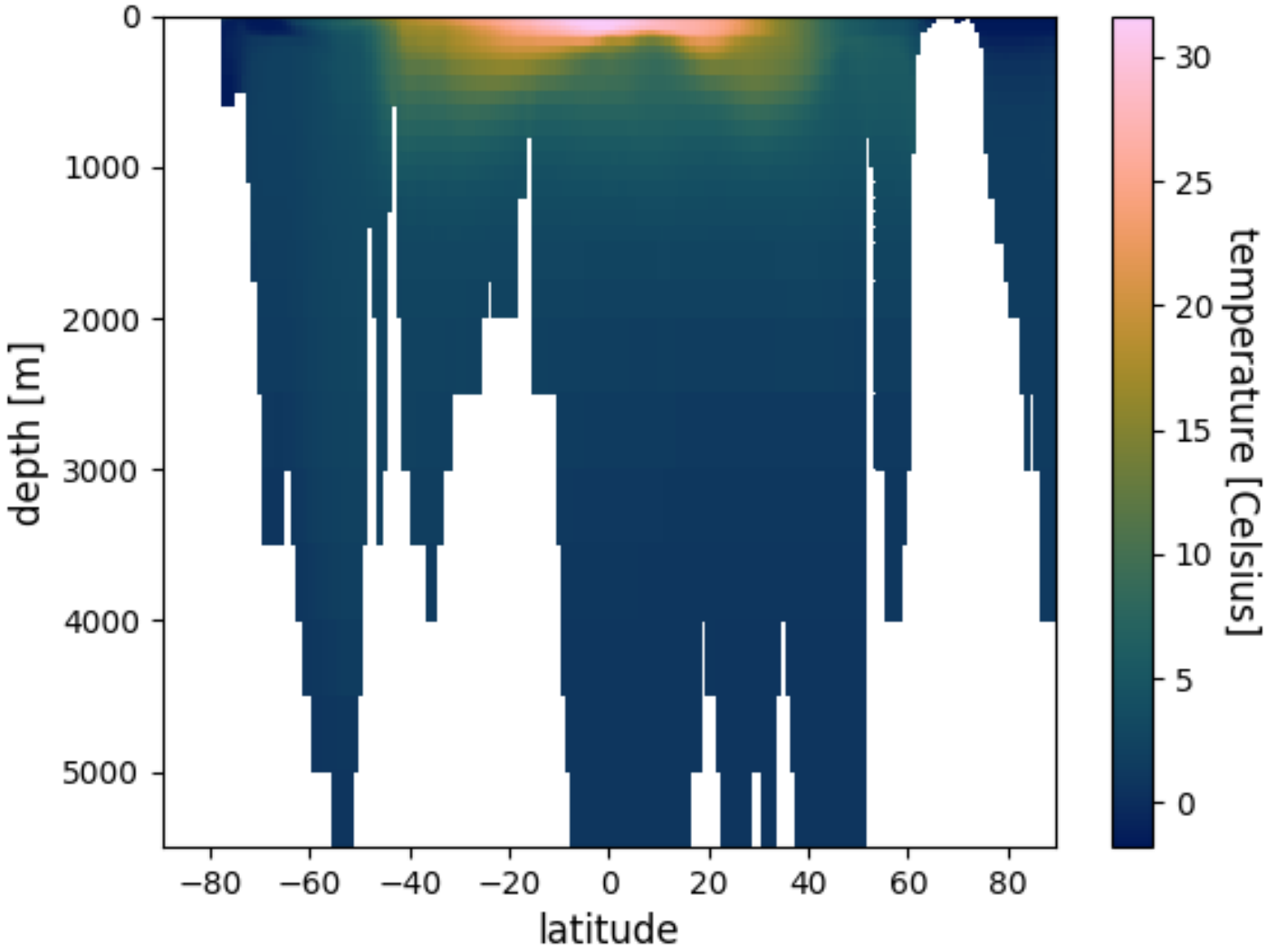
To reverse the plot:

```
# revert axes
ax = plot.axes
ax.invert_yaxis()
```

And add labels, colorbar,...

```
level = ncfile.variables['lev']
lat = ncfile.variables['lat']

# create a mesh grid
Lat,Level=np.meshgrid(lat,level);

#plot
plot=plt.pcolor(Lat,Level,var,cmap=batlow_map)

# add axes labels
plt.xlabel('latitude',fontsize = ftsize)
plt.ylabel('depth [m]',fontsize = ftsize)

# colorbar
cbar = plt.colorbar(plot)
cbar.ax.get_yaxis().labelpad = 15
cbar.ax.set_ylabel('temperature [Celsius]',fontsize = ftsize,rotation=270)
```

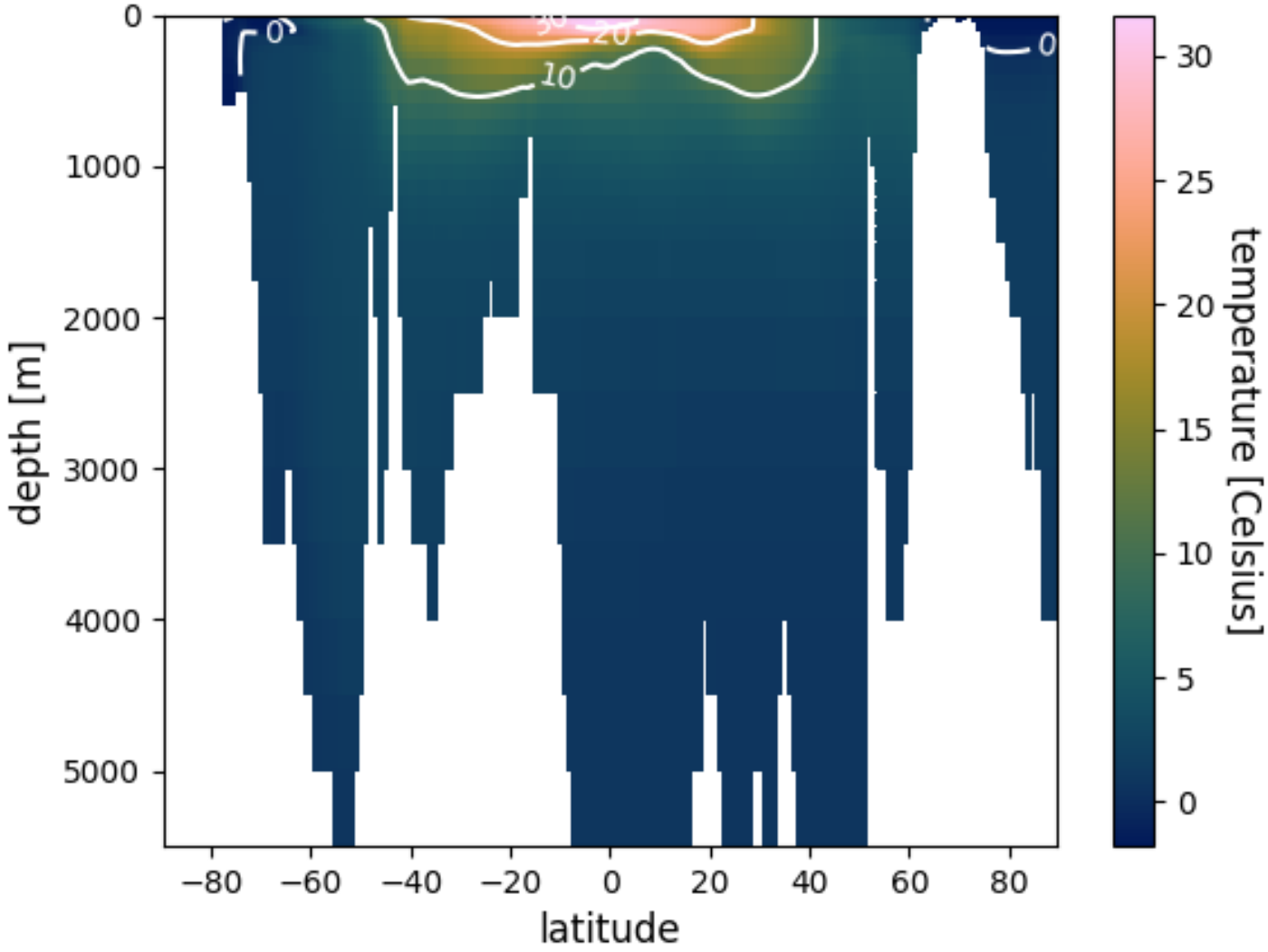
4 | Load into Python and visualise - cross-sections



And finally add contour lines:

```
# plot
plot=plt.pcolor(Lat,Level,var,cmap=batlow_map)

# add contour lines
cont=plt.contour(Lat,Level,var,[0,10,20,30],colors='w')
plt.clabel(cont,fmt='%d',inline=1)
```

4 | Load into Python and visualise - cross-sections



4 | Load into Python and visualise - time-depth diagram

Let's plot one location over time, with depth:

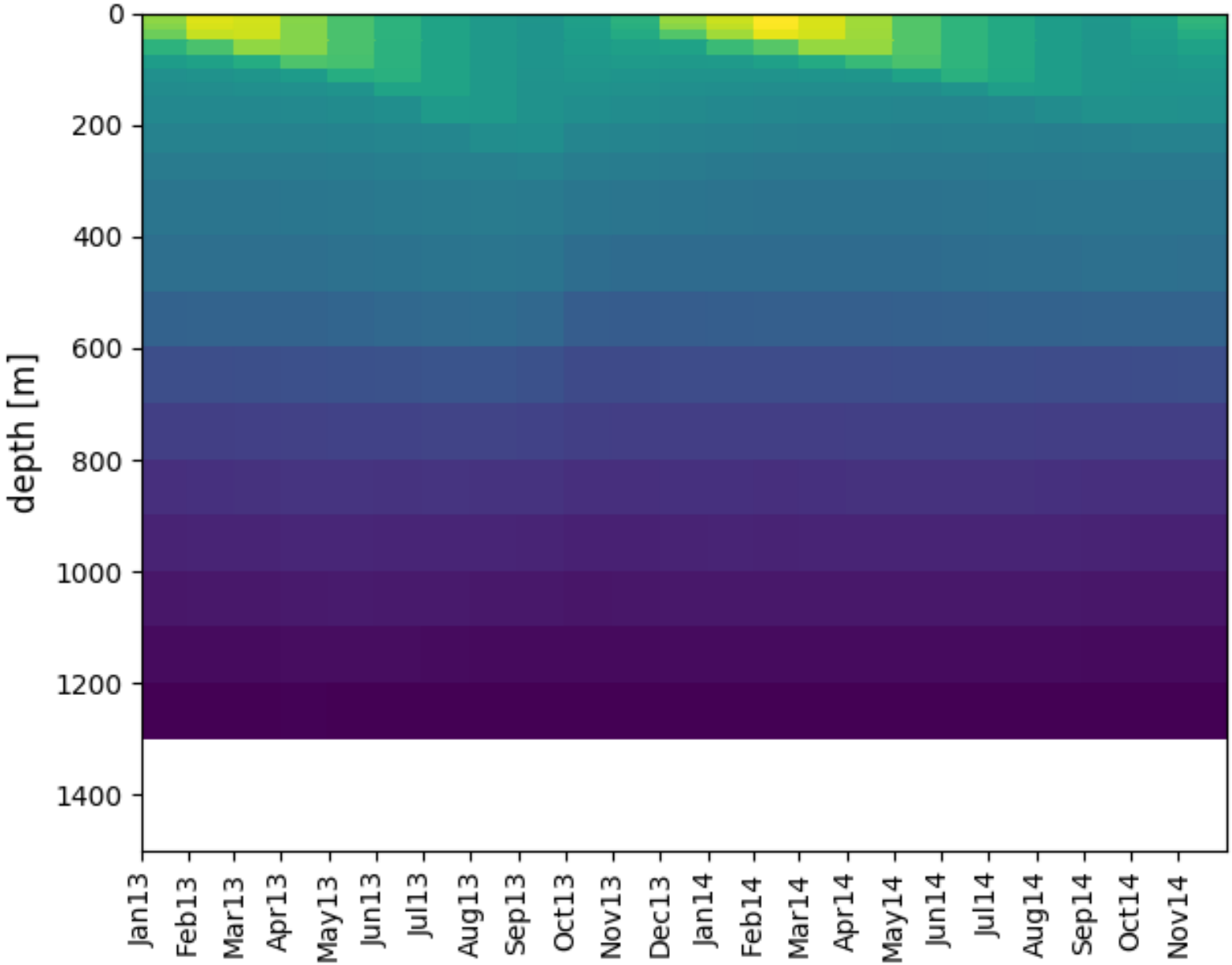
```
# read in the data
filename='thetao_Omon_CESM2_historical_r1i1p1f1_gr_201301-201412.nc'
ncfile = Dataset(filename)

# select one location, all levels and timesteps
var = ncfile.variables['thetao'][:,:,45,180]
level = ncfile.variables['lev']
lat = ncfile.variables['lat']
time = ncfile.variables['time']

# to match time X levels
var=np.transpose(var)

# plot
plot=plt.pcolor(time,level,var)

plt.ylim(0,1500)
ax=plot.axes
ax.invert_yaxis()
```

4 | Load into Python and visualise - time-depth diagram

Let's plot one location over time, with depth:

```
# define time labels
x=(time[0:23])

label=['Jan13','Feb13','Mar13','Apr13','May13','Jun13',
'Jul13','Aug13','Sep13','Oct13','Nov13','Dec13','Jan14',
'Feb14','Mar14','Apr14','May14','Jun14','Jul14','Aug14',
'Sep14','Oct14','Nov14','Dec14']

plt.ylabel('depth [m]',fontsize = ftsize)

# plot time ticks

plt.xticks(x,label,rotation='vertical')
```

4 | Load into Python and visualise - time-depth diagram



4 | Load into Python and visualise - pop quiz

Pop quiz:

- Try to find another location (different hemisphere or equator), or change the depth range on the y-axis.
- Do an horizontal section along the equator / 30° S with depth values. Compare the extent of the mixed layers

Now that we have a pretty figure, let's start being fancy.

We will use the cartopy package:

```
import numpy as np
from netCDF4 import Dataset
from batlow import *    # to have a *fancy* colormap
import matplotlib.pyplot as plt

# tool to plot maps
import cartopy.crs as ccrs
```

4 | Make a publication figure - cartopy

Let's plot SST at the surface, over the whole domain with the 'Plate Carree" projection

```
var = ncfile.variables['thetao'][0,0,:,:]
lat = ncfile.variables['lat']
lon = ncfile.variables['lon']
lev = ncfile.variables['lev']

# get the PlateCarree projection, centered over longitude 0
ax = plt.axes(projection=ccrs.PlateCarree(central_longitude=0.0))

# add coastlines
ax.coastlines(color='grey')

# it is  global file, but show only the southern hemisphere
ax.set_extent([-180, 180, -90, 0], ccrs.PlateCarree())
```

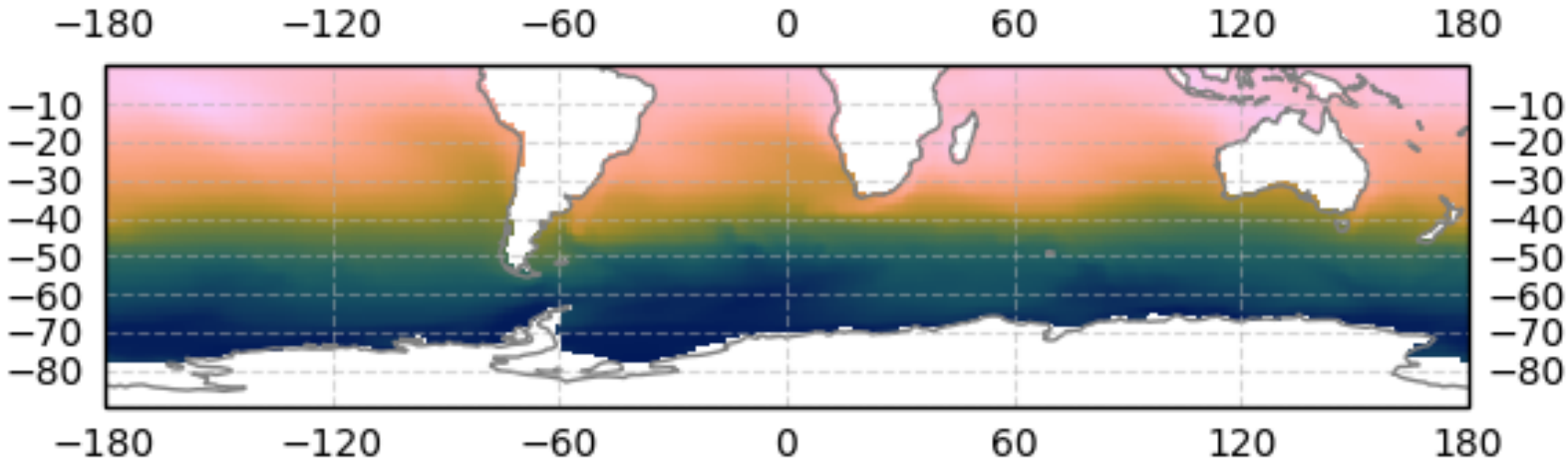
4 | Make a publication figure - cartopy

4 | Make a publication figure - cartopy

Let's fill in the variable, but lat and lon have to be 2D (use meshgrid)

```
# create a mesh grid
Lon,Lat=np.meshgrid(lon,lat);

# plot the variable (+ *fancy* colormap)
plt.pcolor(Lon,Lat,var,transform=ccrs.PlateCarree(),cmap=batlow_map)

gl = ax.gridlines(crs=ccrs.PlateCarree(),
    draw_labels=True,alpha=0.5, linestyle='--')
```

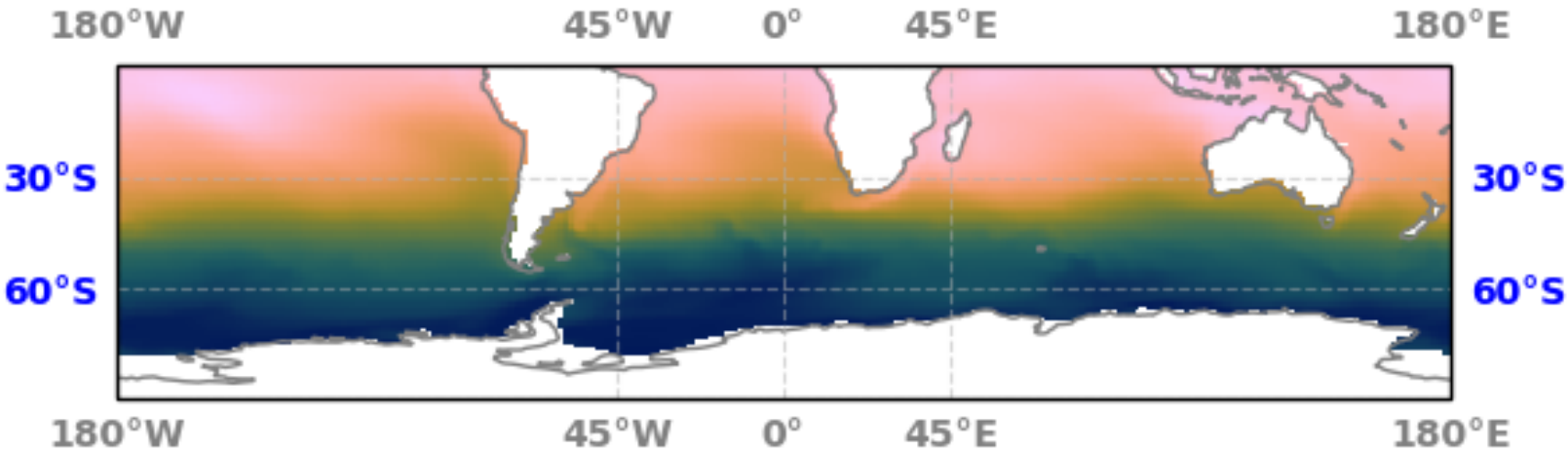
4 | Make a publication figure - cartopy



Let's format the axes:

```
# we need some extra packages:
import matplotlib.ticker as mticker
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER

# add axes labels
gl.xformatter = LONGITUDE_FORMATTER
gl.yformatter = LATITUDE_FORMATTER

# choose where to put the ticks
gl.xlocator = mticker.FixedLocator([-180, -90, 0, 90, 180])
gl.ylocator = mticker.FixedLocator([0,-30, -60, -90])

# customize the font
gl.xlabel_style = {'color': 'grey', 'weight': 'bold'}
gl.ylabel_style = {'color': 'grey', 'weight': 'bold'}
```

4 | Make a publication figure - cartopy



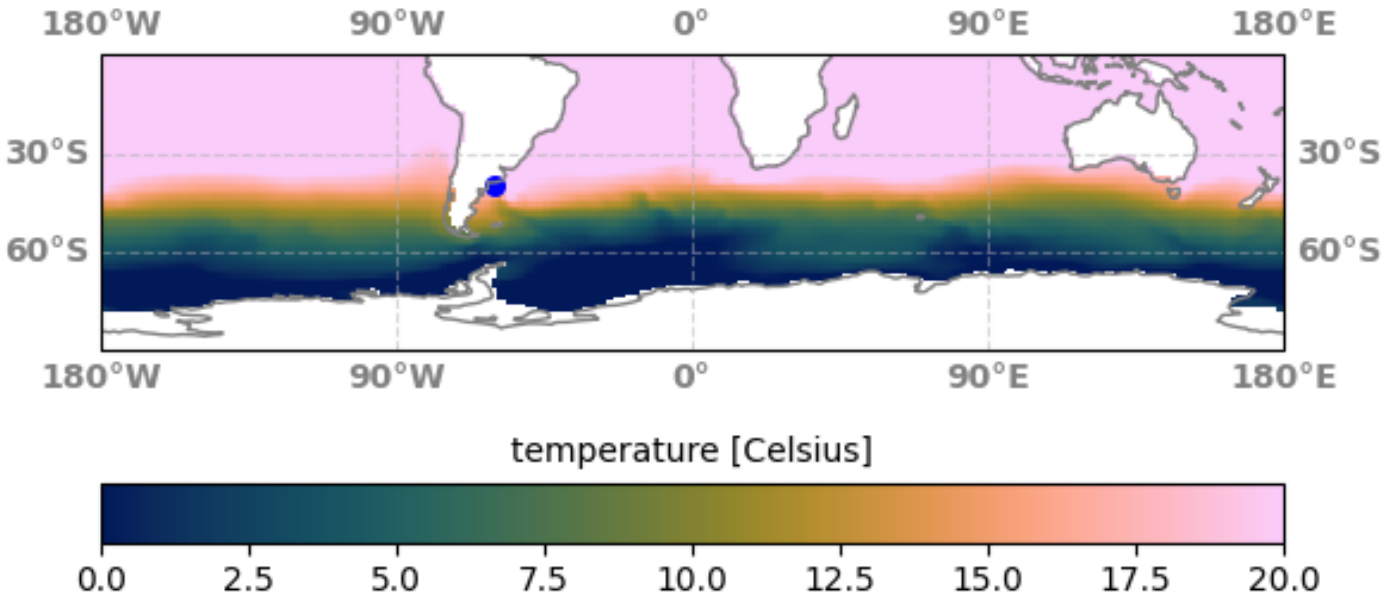
4 | Make a publication figure - cartopy

Imagine:

- we have a CTD station (at 60°CW 40°CS and 4500 m depth (30th level))
- and let's add the colorbar

```
# colorbar
cbar = plt.colorbar(orientation='horizontal')
cbar.ax.get_yaxis().labelpad = 15
cbar.ax.set_title('temperature [Celsius]',fontsize = ftsize)
plt.clim(0,20)

# add the CTD point
plt.scatter(-60,-40,30,color='blue') # 60 W - 40 S
```

4 | Make a publication figure - cartopy



4 | Make a publication figure - **xarray**

xarray is an useful package when dealing with netCDFs containing metadata:

Why xarray?

Multi-dimensional (a.k.a. N-dimensional, ND) arrays (sometimes called "tensors") are an essential part of computational science. They are encountered in a wide range of fields, including physics, astronomy, geoscience, bioinformatics, engineering, finance, and deep learning. In Python, NumPy provides the fundamental data structure and API for working with raw ND arrays. However, real-world datasets are usually more than just raw numbers; they have labels which encode information about how the array values map to locations in space, time, etc.

Xarray doesn't just keep track of labels on arrays -- it uses them to provide a powerful and concise interface. For example:

- Apply operations over dimensions by name: `x.sum('time')`.
- Select values by label instead of integer location: `x.loc['2014-01-01']` or `x.sel(time='2014-01-01')`.
- Mathematical operations (e.g., `x - y`) vectorize across multiple dimensions (array broadcasting) based on dimension names, not shape.
- Flexible split-apply-combine operations with groupby: `x.groupby('time.dayofyear').mean()`.
- Database like alignment based on coordinate labels that smoothly handles missing values: `x, y = xr.align(x, y, join='outer')`.
- Keep track of arbitrary metadata in the form of a Python dictionary: `x.attrs`.

```
# xarray
import xarray as xr

ftsize=10

filename='../data/thetao_Omon_CESM2_historical_r1i1p1f1_gr_20
data=xr.open_dataset(filename)

print(data)
```

4 | Make a publication figure - **xarray**

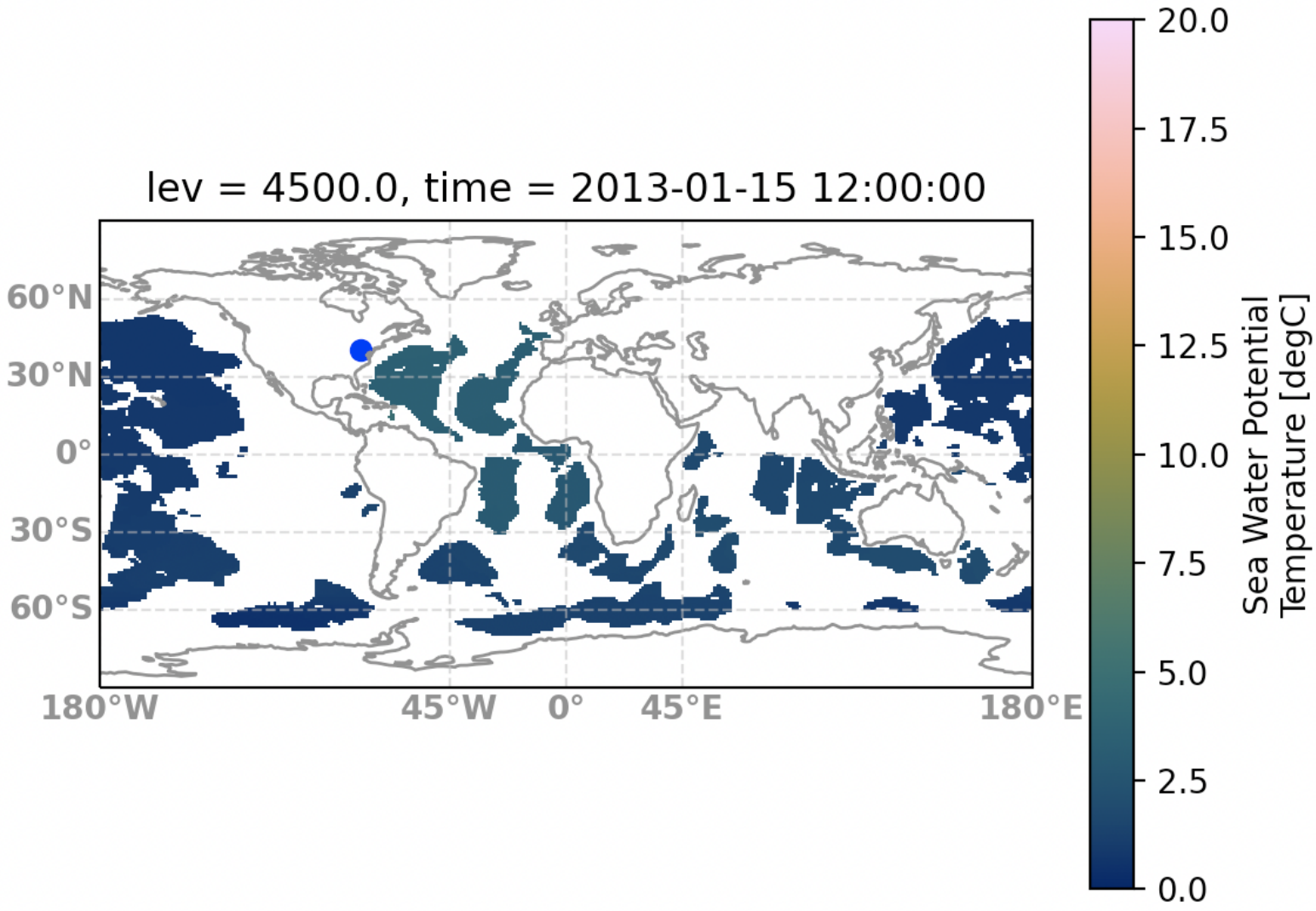
```
<xarray.Dataset>
Dimensions:    (lat: 180, d2: 2, lev: 33, lon: 360, time: 24)
Coordinates:
  * lat        (lat) float64 -89.5 -88.5 -87.5 -86.5 ... 86.5 87.5 88.5 89.5
  * lev        (lev) float64 0.0 10.0 20.0 30.0 ... 4e+03 4.5e+03 5e+03 5.5e+03
  * lon        (lon) float64 0.5 1.5 2.5 3.5 4.5 ... 356.5 357.5 358.5 359.5
  * time       (time) object 2013-01-15 12:00:00 ... 2014-12-15 12:00:00
Dimensions without coordinates: d2
Data variables:
    lat_bnds   (lat, d2) float64 ...
    lev_bnds   (lev, d2) float64 ...
    lon_bnds   (lon, d2) float64 ...
    thetao     (time, lev, lat, lon) float32 ...
    time_bnds  (time, d2) object ...
Attributes: (12/47)
    Conventions:            CF-1.7 CMIP-6.2
    activity_id:            CMIP
    case_id:                15
    cesm_casename:          b.e21.BHIST.f09_g17.CMIP6-historical.001
    contact:                cesm_cmip6@ucar.edu
    creation_date:          2019-01-16T22:01:19Z
    ...                     ...
    branch_time_in_child:   674885.0
    branch_method:          standard
    further_info_url:       https://furtherinfo.es-doc.org/CMIP6.NCAR.CESM2.h...
    sub_experiment:         none
    history:                Wed Aug  4 15:18:51 2021: ncks -d time,1956,1979 ...
    NCO:                    4.7.2
```

4 | Make a publication figure - **xarray**

```
theta0=data.thetao.isel(time=0,lev=30)
point =data.thetao.sel(time="2013-01",lat=-79.5,lon=40.5,lev=400)

# get the PlateCarree projection, centered over longitude 0
ax = plt.axes(projection=ccrs.PlateCarree(central_longitude=0.0))
ax.coastlines(color='grey')
ax.set_global()

theta0.plot.pcolormesh(ax=ax,transform=ccrs.PlateCarree(), x='lon',
    y='lat', cmap=batlow_map,vmin=0,vmax=20)

# add the CTD point
plt.scatter(point.lat,point.lon,color='blue') # 60 W - 40 S
```

4 | Make a publication figure - **xarray**

The title, colorbar,.. are automatically added:



4 | Make a publication figure - difference between two datasets in **cdo**

What if we want to compare two datasets, which are not on the same grid?

Re-Mapping is the answer !

We can do it in **cdo**:
cdo remapbil ... will use bilinear interpolation to remap one grid ont another
(! create a grid description file using *cdo griddes* beforehand)

4 | Make a publication figure - difference between two datasets in **cdo**

Get the grid description of the file you want to remap **onto**: (here, ERA5)

cdo griddes SST_ERA5_201301.nc > grid_era5.txt

```
#
# gridID 1
#
gridtype  = lonlat
gridsize  = 1038240
xsize     = 1440
ysize     = 721
xname     = longitude
xlongname = "longitude"
xunits    = "degrees_east"
yname     = latitude
ylongname = "latitude"
yunits    = "degrees_north"
xfirst    = 0
xinc      = 0.25
yfirst    = 90
yinc      = -0.25
scanningMode = 64
```

4 | Make a publication figure - difference between two datasets in **cdo**

Then:

- we remap using a bilinear interpolation:
  cdo remapbil,grid_era5.txt
  thetao_Omon_CESM2_historical_r1i1p1f1_gr_201301-201412.nc theta_remapped.nc

- to subtract one from the other, the two files have to have the 'same variable' (name):
  cdo chname,thetao,sst theta_remapped.nc theta_remapped_sst.nc

- and subtract the two files:
  cdo sub SST_ERA5_201301.nc theta_remapped_sst.nc diff_era5_theta.nc

- let's check the nc file, and play with the colorbar, range,...

4 | Make a publication figure - remap with griddata

Now, let's do the same in Python !

We will use the **griddata package**

```
# interpolation tool
from scipy.interpolate import griddata

# load  data
filename_cmip='thetao_Omon_CESM2_historical_r1i1p1f1_gr_201301-201412.nc'
filename_era5='SST_ERA5_201301.nc'

ncfile_cmip = Dataset(filename_cmip)
ncfile_era5 = Dataset(filename_era5)

var_cmip = ncfile_cmip.variables['thetao'][0,0,:,:]
lat_cmip = ncfile_cmip.variables['lat']; lat_cmip=np.array(lat_cmip)
lon_cmip = ncfile_cmip.variables['lon']; lon_cmip=np.array(lon_cmip)

var_era5= ncfile_era5.variables['sst'][0,:,:]-273.15
lat_era5 = ncfile_era5.variables['latitude']; lat_era5=np.array(lat_era5)
lon_era5 = ncfile_era5.variables['longitude'];lon_era5=np.array(lon_era5)
```

4 | Make a publication figure - remap with griddata

And it needs the data to be formatted in a specific way:

```
# the latitude/longitude variables are 1D and we need 2D: use meshgrid

grid_lon_era5,grid_lat_era5 = np.meshgrid(lon_era5,lat_era5)
grid_lon_cmip,grid_lat_cmip = np.meshgrid(lon_cmip,lat_cmip)

# the latitude/longitude coordinates should be in the
# shape(2,N): use ".ravel()" to get all the points from 2D to 1D
# and "stack" to glue them together

points_cmip= np.stack([grid_lat_cmip.ravel(), grid_lon_cmip.ravel()], -1)
```

And it needs the data to be formatted in a specific way:

```
# now we can interpolate, using griddata
# the input is
#  - points: the (2,N) grid of points to interpolate
#  - theta: the values to interpolate
#       (as one long array, using .ravel())
#  - (grid_latera,grid_longera): the two 2D coordinates
#        of the grid we interpolate into

var_cmip_regridded = griddata(points_cmip, var_cmip.ravel(),
(grid_lat_era5,grid_lon_era5),method='nearest')
```

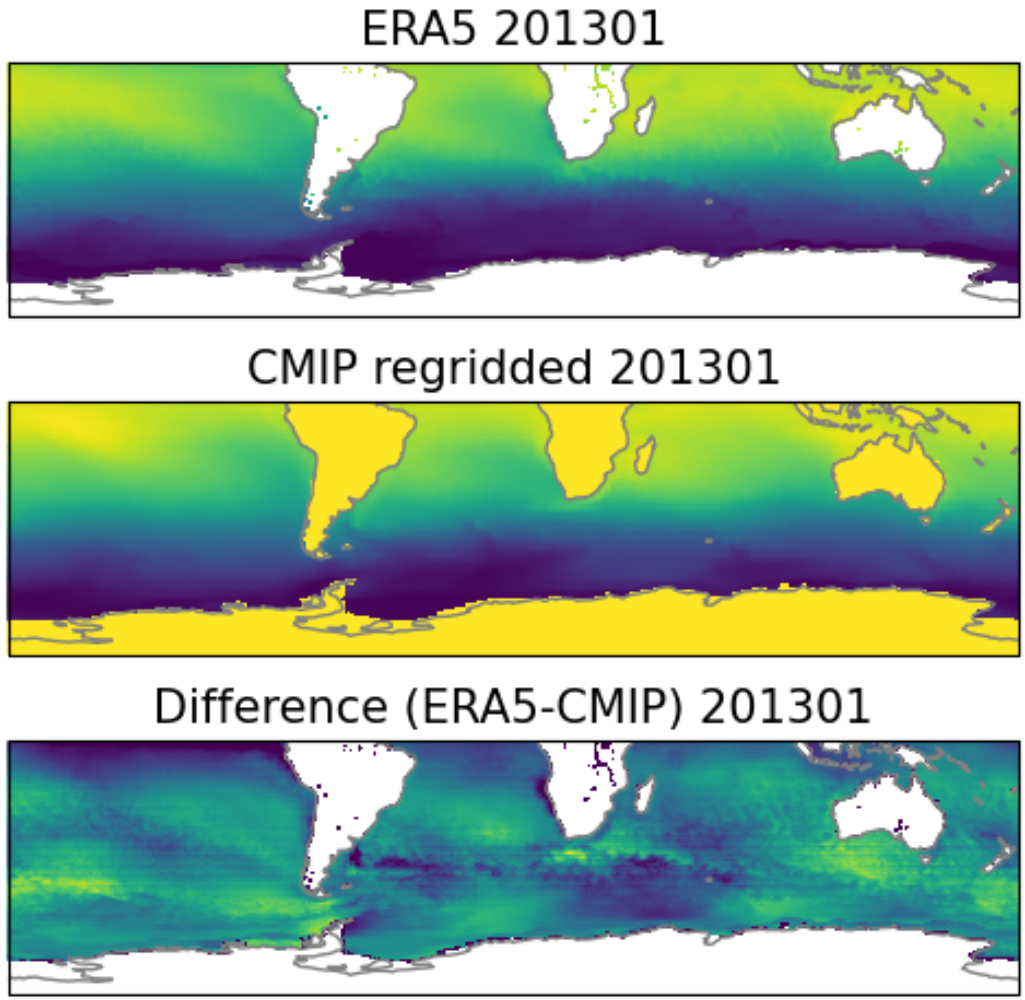
4 | Make a publication figure - make different subplots

Let's make 3 subplots in the figure:

- the ERA5 data
- the CMIP regridded data
- the difference between the two

```
# plot using subplots:

fig=plt.figure()

A=fig.add_subplot(3,1,1,projection=ccrs.PlateCarree())
A.coastlines(color='grey')
A.set_extent([-180,180,-90,0],ccrs.PlateCarree())
plt.title('ERA5 201301',fontsize=ftsize)
plt.pcolor(grid_lon_era5,grid_lat_era5,var_era5, transform=ccrs.PlateCarree())
```

4 | Make a publication figure - make different subplots



4 | Make a publication figure - make different subplots

Add the colorbars :

- batlow, but same limits as the second subplot
- batlow, but same limits as the first subplot
- red-blue (the vik colorbar)!

Hint: we can print out the min and max values of our variables :

```
print(np.min(var_era5))
```

4 | Make a publication figure - make different subplots

```
A=fig.add_subplot(3,1,1,projection=ccrs.PlateCarree())
A.coastlines(color='grey')
A.set_extent([-180,180,-90,0],ccrs.PlateCarree())
plt.title('ERA5 201301',fontsize=ftsize)
plt.pcolor(grid_lon_era5,grid_lat_era5,var_era5,
    vmin=-2,vmax=33,transform=ccrs.PlateCarree(),cmap=batlow_map)
cbar = plt.colorbar()
cbar.ax.get_yaxis().labelpad = 15
cbar.ax.set_ylabel('SST [\xb0 C]',fontsize = ftsize2,rotation=270)
```

And to center the vik colorbar around 0 :

```
from vik import *
import matplotlib.colors as colors


# function to center around zero:

class MidpointNormalize(colors.Normalize):
    def __init__(self, vmin=None, vmax=None, midpoint=None, clip=False):
        self.midpoint = midpoint
        colors.Normalize.__init__(self, vmin, vmax, clip)

    def __call__(self, value, clip=None):
        # I'm ignoring masked values and all kinds of edge cases to make a
        # simple example...
        x, y = [self.vmin, self.midpoint, self.vmax], [0, 0.5, 1]
        return np.ma.masked_array(np.interp(value, x, y), np.isnan(value))
```

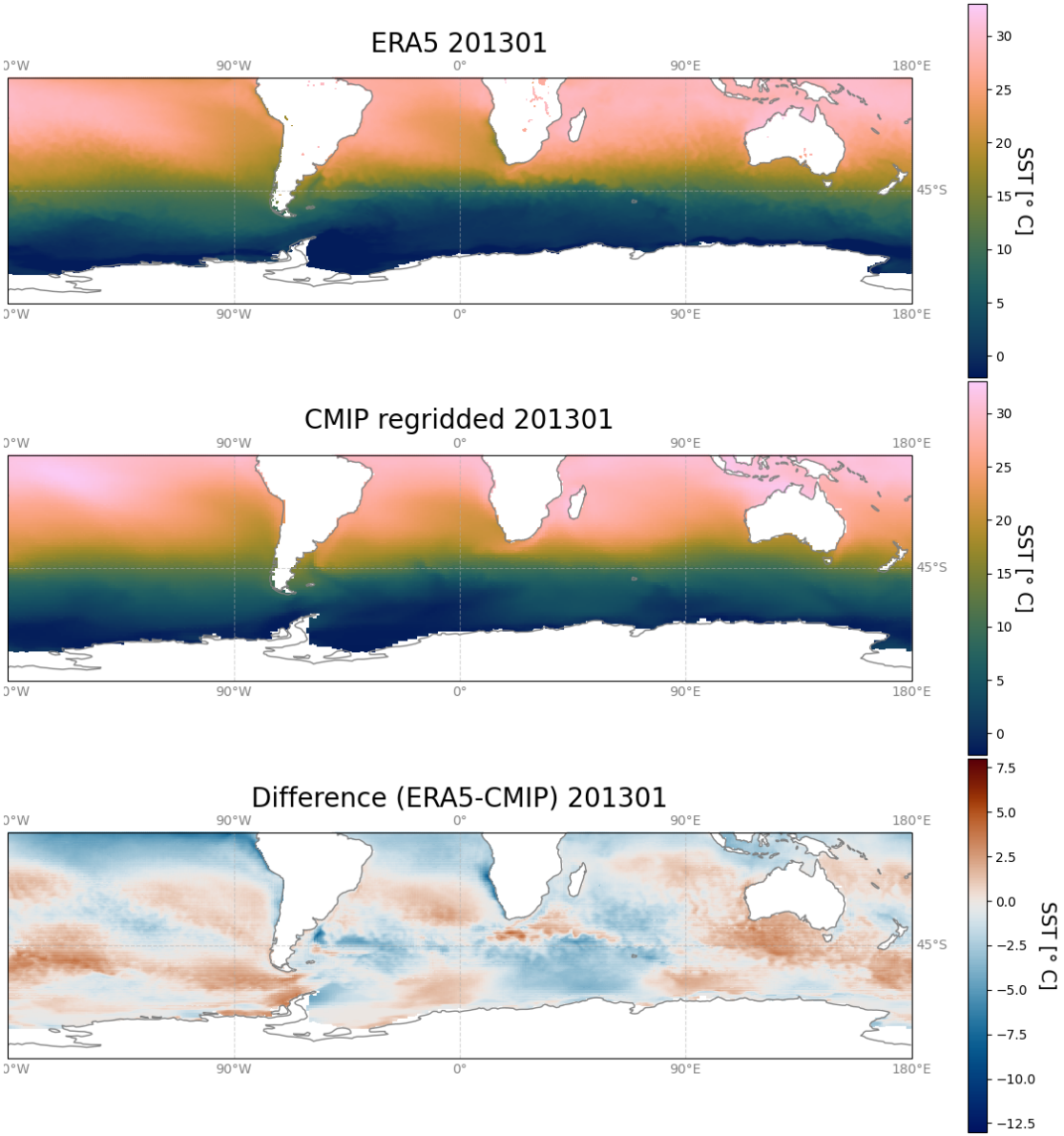
4 | Make a publication figure - make different subplots

And to center the vik colorbar around 0:

(hint: find out the min and max values)

```
# add the min, max and centre values for the blue-red colorbar:
mid_val=0; minvar = -13; maxvar = 8;
# plot using the limits
plt.pcolor(grid_lon_era5,grid_lat_era5,var_era5-var_cmip_regridded,
    transform=ccrs.PlateCarree(),cmap=vik_map,
    norm=MidpointNormalize(midpoint=mid_val,vmin=minvar, vmax=maxvar))
cbar = plt.colorbar()
cbar.ax.get_yaxis().labelpad = 15
cbar.ax.set_ylabel('SST [\xb0 C]',fontsize = ftsize2,rotation=270)
```

4 | Make a publication figure - make different subplots



Pop quiz:
Our figure will **not** look like this, fin

4 | Make a publication figure - save the figure

Finally, let's set the layout and save the figure:

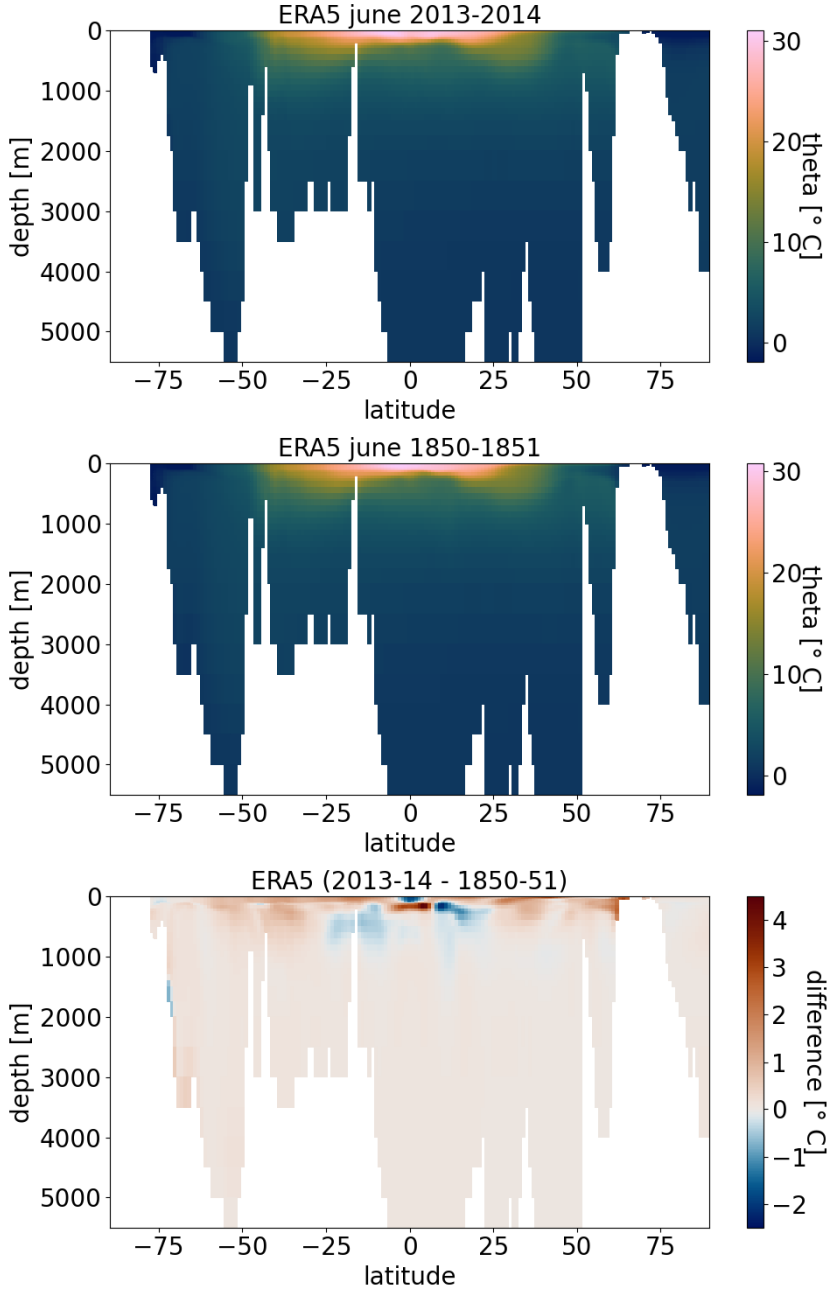
```
fig=plt.figure(figsize=(12,12))

fig.tight_layout(pad=0.3)

plt.savefig('ERA5_CMIP_difference_sst_201301.png')
```

4 | Make a publication figure - pop quiz

Pop quiz:
Let's do the same as for the final *cdo* command, but in a python script:

- two cross sections
- of mean June
- sea surface temperature
- over the dateline
- using the two thetao files (201301-201412 and 185001-185112)
- using the red-blue colormap
- with the surface at the top
- and a colorbar symmetrical around 0

4 | Make a publication figure - pop quiz

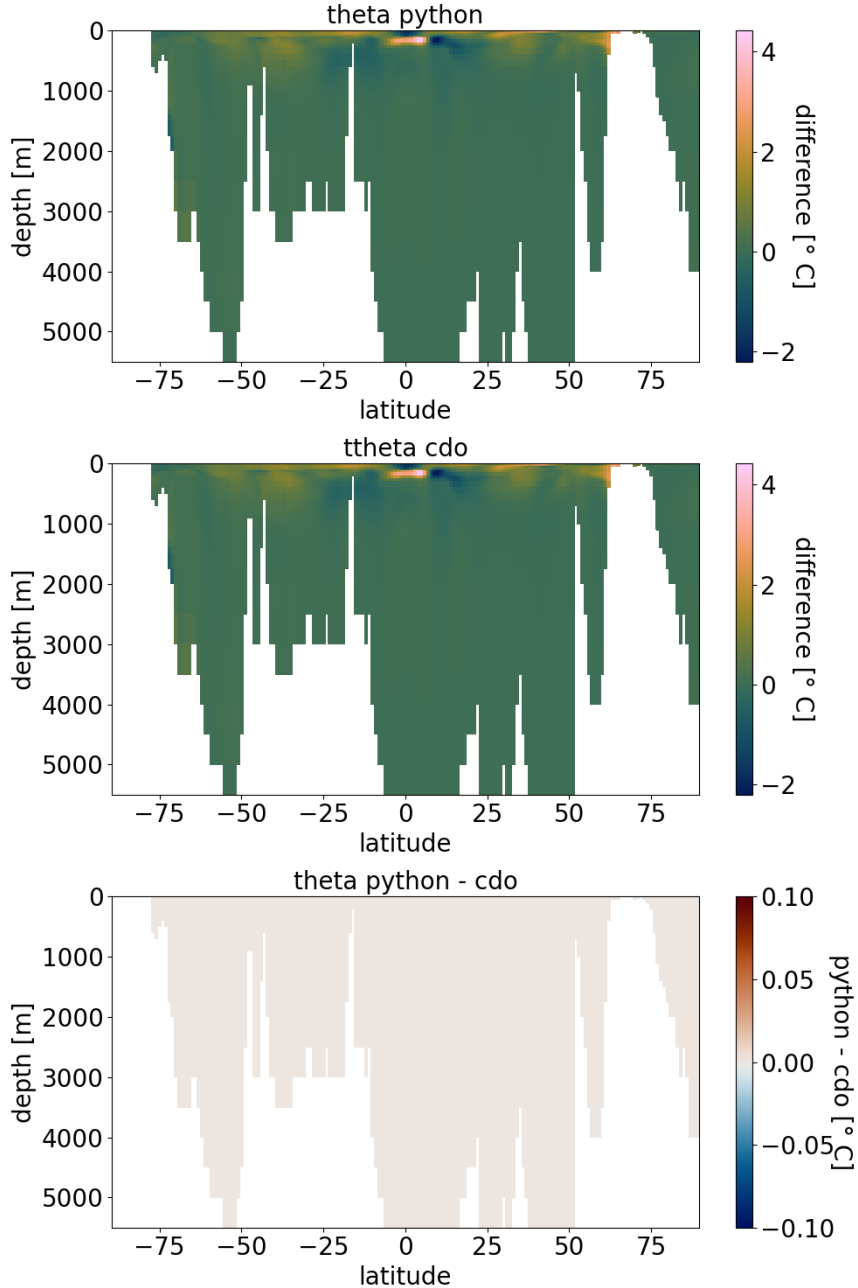


4 | Make a publication figure - pop quiz

And as a final check,

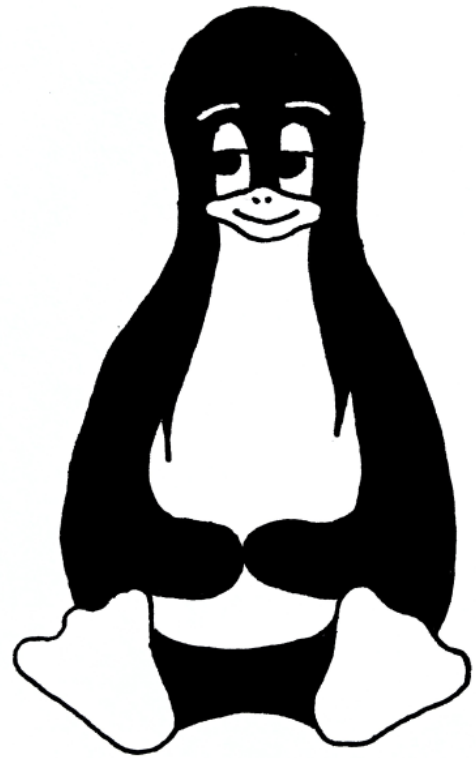
load your cdo final result and

compare to the output of this script!



6 | Outline

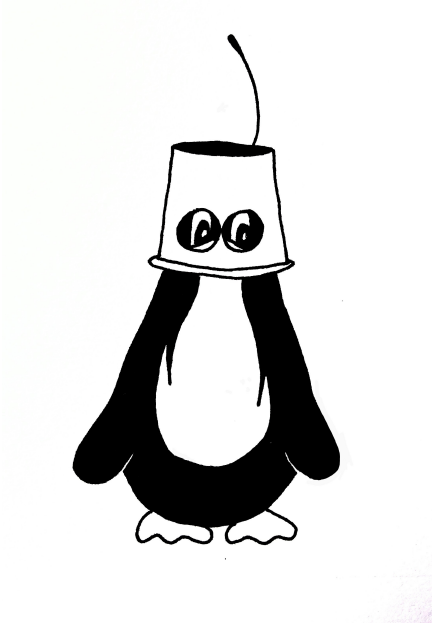
- Reality Check
- Workflow
- Work Structure
- Folder Structure
- Version Control
- File Naming
- Data Handling
- Best Practise



6 | Reality Check

I plotted this real nice figure for Richard a while ago. Now Peter wants something very similar. The graph just needs a different colormap and smaller units. I wish I used a damn script and saved it.

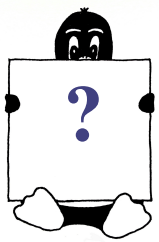
I wrote this really flash paragraph about extra-terrestian fossils in my samples. Rob thought it was bogus. Now that they have landed in Area 51 to collect their trash I would really like that paragraph back in the manuscript.



6 | Reality Check

Finally I got around to publish some of that ancient thesis material from 5 years ago. Just received an email from the journal's editor, requesting a re-plot of figures in high res.

Lockdown was fun - got lots of processing and writing done. Will copy and backup files when I get back to the office next week.
...
My labtop dropped off the table last night. The hard drive is jammed.



* **How do you work?**
* **What is your workflow?**
* **Does your work structure map your workflow?**

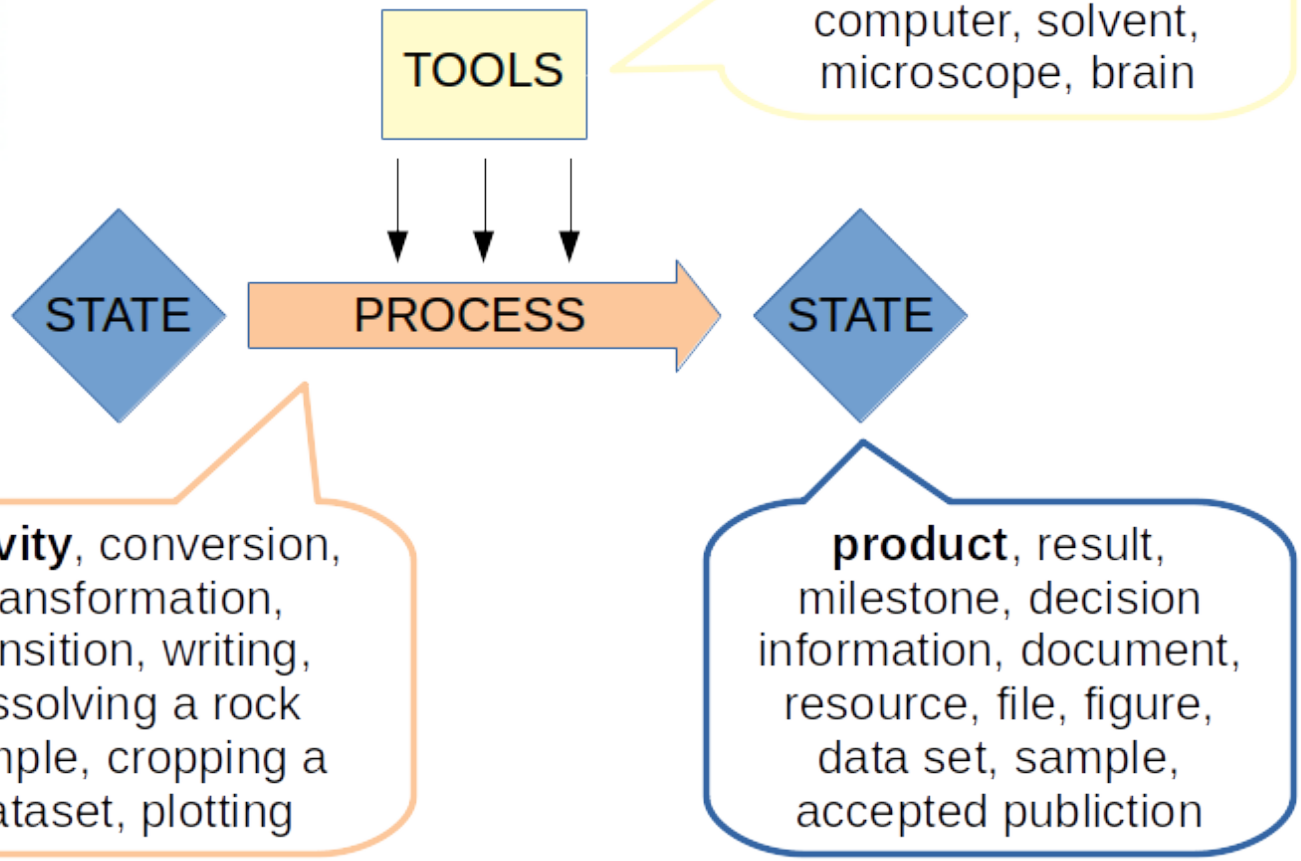
6 | Workflow

*"A workflow consists of an orchestrated and **repeatable** pattern of activity, enabled by the systematic organization of resources into processes that transform matericals, provide services, or process information."* (Wikipedia)

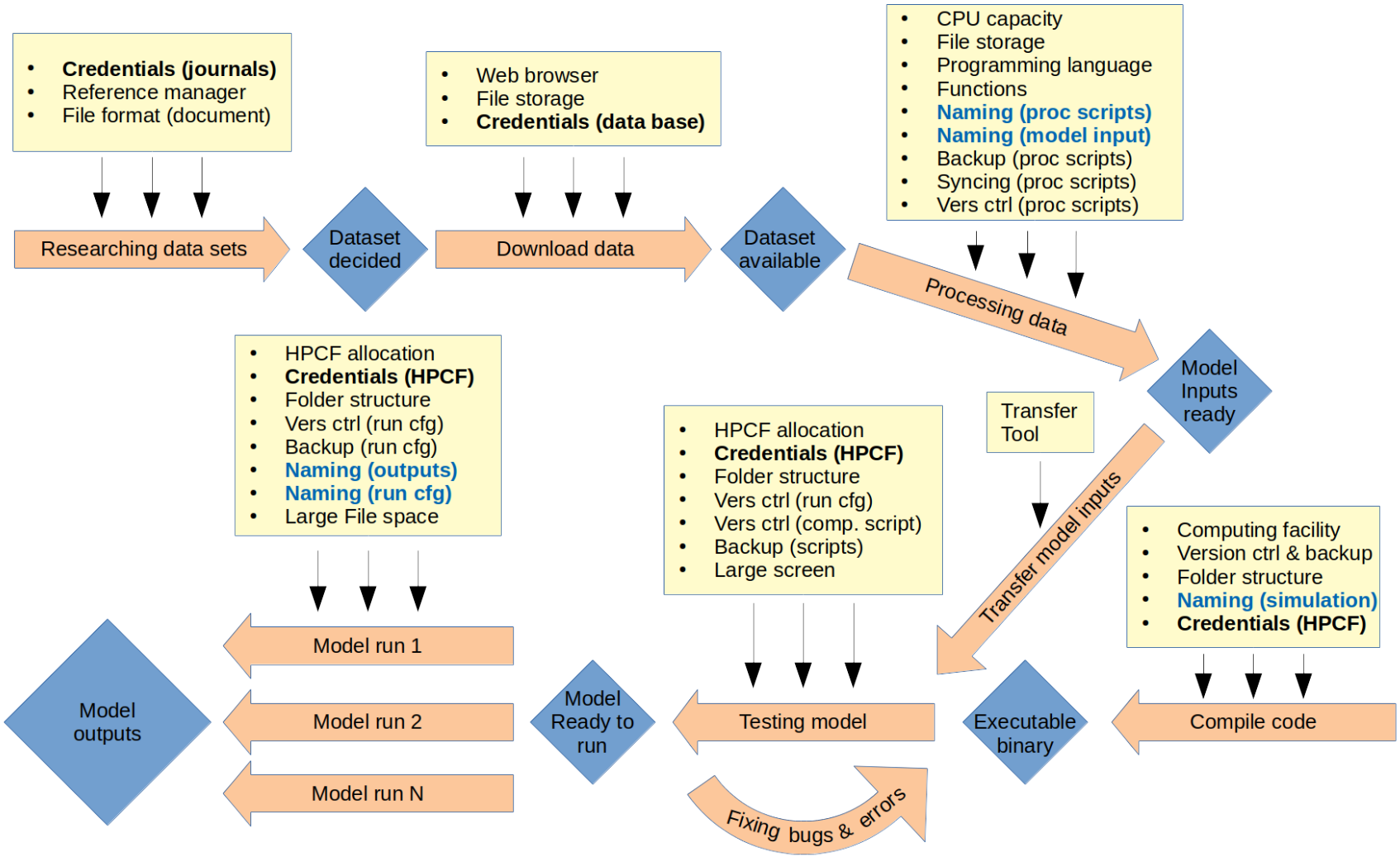
- Examples
  - All physical and mental steps between your hypothesis and the published result.
  - All related tasks and stages to carry out an experiment.
  - All activities, materials and documents to measure a rock sample.

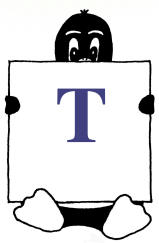
- **Why do we need to think about our workflows?**
- Repeatability and reproducibility are key requirements in science. For example
  - Result of a numerical model.
  - Chemical analysis of a rock sample.
  - Pre-processing of a dataset.
  - A figure in a publication.
- Maximizes work efficiency and increases effectiveness.

6 | Workflow - concept & terminology



6 | Workflow - example





* Think of a small project in the recent past and sketch the associated workflow. Examples could be: writing a Marsden one pager, set up and run a simulation or analyse rocks and record and store the data. Use the concept of State, Process, Tool

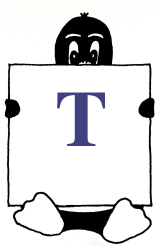
6 | Work Structure

*A suitable **work structure** is a systematically organized work environment that **maps** one specific or a group of **workflows**.*

- The guiding principles to arrange the environment into a suitable structure are
  - Reproducability
  - Efficiency
  - Cost - in units of storage, processing time, computing load, money, FTE
- For computer based environments the work structure typically consists of blocks such as

- data sets
- software & apps
- compilers & data language
- **file formats**
- **version control**
- computer hardware
- storage facilities
- **backup procedure**
- communication model
- **folder structure**
- exchange & transfer protocol
- documentation
- **data handling**
- credential management

- A specific work structure consists of a subset of these blocks. They are not absolute definitions and contain a deliberate degree of redundancy.
- **The list of building blocks can be used as a stencil to filter out the important aspects in the workflow that need to be incorporated in your work structure.**



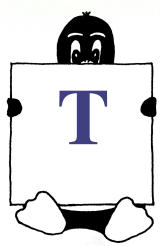
∗ Find two blocks that are important for your work structure and one that is negligible.

6 | Work Structure - discussion

- The implementation of a Work Structure can range from installing a plugin in your browser to manage all your online credentials to writing a large bash scripts that automatically synchronize all your Python scripts between your office desktop and your labotp at home.
- Apart from the three guiding principles there are other motivations for a well organized work structure
  - the frustration having to recreate something that has been "lost"
  - can not afford to loose a good idea
  - wasting time to search for something
- Protect the products of your creativity, things that were made with your intellectual powers.
- Even a costly computer model could be re-run. A good idea that you did not write down might not come back to you for any money. (How to create ideas is another lecture.)
- There is no right or wrong work structure. There is one or none.
- **Feedback** - the process of tayloring your work structure around your own wokflow helps you streamlining the workflow itself.

6 | Folder Structure

- Arguably an effective folder structure is the most important and most visible component of a work structure.
- There are multiple ways to sort / categorize files.
  - most commonly folders map the chronological order of tasks or task groups
  - grouping by file types
  - theme based
- A good folder structure should
  - map the workflow
  - contain no more than 6 to 10 subfolders for quick navigation
  - have names that are self explaining
  - separate data from tools
  - separate backed up items from those that are not
  - enables selective syncing with cloud services



∗ Design a future proof folder structure for your publications that matches your computer based workflow.

- Example modelling folder

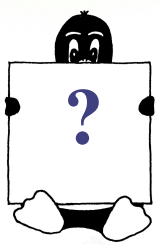
```
027
├── 01_preparation
│   ├── pre_data
│   └── scripts
│       └── arc
├── 02_input
├── 03_simulation
│   ├── SIM_NAME_A
│   ├── SIM_NAME_B
│   └── SIM_NAME_C
├── 04_output
└── 05_processing
    ├── figures
    ├── movies
    ├── pro_data
    └── scripts
        └── arc
028
├── 01_preparation
│   ├── pre_data
│   └── scripts
│       └── arc
├── 02_input
├── 03_simulation
│   ├── SIM_NAME_A
│   ├── SIM_NAME_B
│   └── SIM_NAME_C
├── 04_output
└── 05_processing
    ├── figures
    ├── movies
    ├── pro_data
    └── scripts
        └── arc
```

6 | Version Control - not a **How-To-Do** rather a **Do-NoMatter-How!**

- There are many reasons why one has to go back to a previous version of something.
- Do not ever delete or overwrite something that is a result of your intellectual powers (IP)
- Freeing disk space is no longer a justified reason to delete. Even a 5000 line script consumes less than 200kB and available storage capacity is nowadays on the order of 10^6 times bigger than that.

6 | Version Control - not a **How-To-Do** rather a **Do-NoMatter-How!**

- One option is to use a Version Control System (VCS)
  - allows collaborators to work on the same project
  - prevents overwriting of changes
  - maintains a history of changes
  - manages large repositories of files
  - Distributed VCS are also a backup system
- **Centralized VCS**
  - uses a central server for all files
  - enables team collaboration
  - vulnerable for single point of failure (disk of the central server)
  - requires a separate backup workflow
  - low risk of divergence
- **Distributed VCS**
  - uses a central server and local clients
  - clients mirror the full repository (implicitly failsafe)
  - works offline
  - risk of divergence is mitigated with sophisticated merging algorithms



∗ What are you using for version control?

6 | Version Control | git - stage & commit

- clone the repository

```
git clone https://github.com/sj-science/WinterSchool_train_repo
```

- descend into project directory

```
cd /WinterSchool_train_repo
```

- list the folder content, -l gives more information, -h makes the file size human readable

```
ls -lh
```

  * make a personal copy of the bash script, name it [your_initials]_script
  * execute the bash script

- check the status of your local repository

```
git status
```

- stage new file(s) for commit

```
git add your_initials_script
```

- check status again

```
git status
```

- commiting the change to the local repository with a meaningful description

```
git commit -m 'adding a bash script that says "hello world"'
```

- modify the file

```
vi sj_script
```

- create a new print message

```
echo "World says Hi"
```

- exit editor

```
Esc : x
```

- let's create another file

```
touch empty_file.txt
```

- check status (git lists new and modified files separately)

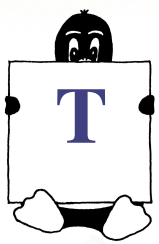
```
git status
```

- *stage* the modified file (only staged files can be committed to the repository)

```
git add sj_script
```

- *commit* the modification & check status (unstaged changes are not effected by the commit)

```
git commit -m "changed the print out message"  git status
```



* modify the print script again, add a second line to the printed message
* stage and commit the modified script and the new file to the repository

- check status

```
git status
```

- display list of commits

```
git reflog
```

- display more details

```
git log
```

- display the script

```
cat sj_script
```

- display the script using git

```
git show [commitID1]:path/sj_script
```

- display the script before the last commit

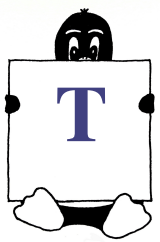
```
git show [commitID2]:path/sj_script
```

- show difference in the script between two commits

```
git diff [commitID1]:path/sj_script [commitID2]:path/sj_script
```

- show all differences between two commits

```
git diff [commitID1] [commitID2]
```



* rename the file empty_file.txt to [your_initials]_file.txt
* edit the renamed file and add some text
* stage and commit the change
* display the difference between your current (latest) and your first commit (hello world)

6 | Version Control | git - rollback

- check status

```
git status
```

- show the commit history of the pointer *HEAD*

```
git reflog
```

- **soft** rollback (both options equivalent)

```
git reset --soft HEAD@{[N]}
git reset --soft [commitID]
```

- **changes are retained as staged, unstaged files are retained**

```
git status
```

- list files

```
ls -l
```

- check the pointer

```
git reflog
```

- **hard** rollback

```
git reset --hard [commitID]
```

- **changes are discarded, unstaged files are retained**

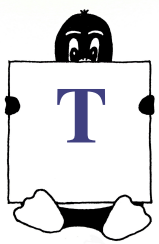
```
git status
```

- list files

```
ls -l
```

- check the pointer

```
git reflog
```



∗ move pointer *HEAD* to the most advanced commit

6 | Version Control | git - push & pull

- check status

`git status`

```
Your branch is ahead of 'origin/play_branch_sj' by 3 commits.
(use "git push" to publish your local commits)
```

- says we can fast forward with a simple command - GREAT!
- push changes to the online repository

`git push`

```
To https://github.com/sj-science/WinterSchool_train_repo
! [rejected] play_branch_sj -> play_branch_sj (fetch first)
error: failed to push some refs to 'https://github.com/sj-science/WinterSchool_train_repo
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

- there is no such thing as "simultaneous" work on a central project / repository / document
- modifications need to be applied sequentially to avoid divergence
- even google.docs commits changes from collaborators sequentially - but very fast

- Two choices - merge or new branch ...
- merge

```
git pull
```

```
 GNU nano 2.9.3 /home/..../.git/MERGE_MSG
Merge branch 'main' of https://github.com/sj-science/WinterSchool_train_repo into main

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

- the merging happens on the local repository

```
ls
```

- remote change has been pulled to local folder

- the file you originally committed has been uncommitted

`git status`

```
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 2 different commits each, respectively.
 (use "git pull" to merge the remote branch into yours)

All conflicts fixed but you are still merging.
 (use "git commit" to conclude merge)

Changes to be committed:
new file: blub.txt
```

- re-commit your new file

```
git commit -m "re-commit my new file to finish merge"
git status
```

```
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

```
git push
git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

git push

```
To https://github.com/sj-science/WinterSchool_train_repo
! [rejected] play_branch_sj -> play_branch_sj (fetch first)
error: failed to push some refs to 'https://github.com/sj-science/WinterSchool_train_rep
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

git branch

```
main
```

- create a new branch

git checkout -b sj_branch

```
Switched to a new branch 'sj_branch'
```

git branch

```
 main
*sj_branch
```

6 | Version Control | git - branch

- check new branch

```
git status; echo; ls
```

```
On branch sj_branch
nothing to commit, working tree clean

sj_script.txt
```

- add the new branch to the remote repository

```
git push -u origin sj_branch
```

```
branch
Counting objects: 12, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (12/12), 1.40 KiB | 1.40 MiB/s, done.
Total 12 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), done.
remote:
remote: Create a pull request for 'sj_branch' on GitHub by visiting:
remote: https://github.com/sj-science/WinterSchool_train_repo/pull/new/sj_branch
remote:
To https://github.com/sj-science/WinterSchool_train_repo.git
 * [new branch] sj_branch -> sj_branch
Branch 'sj_branch' set up to track remote branch 'sj_branch' from 'origin'.
```

git status

```
On branch sj_branch
Your branch is up to date with 'origin/sj_branch'.

nothing to commit, working tree clean
```

- your colleague can now see and access (checkout) your new branch

git fetch

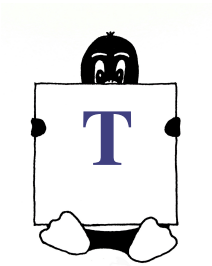
```
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 1), reused 1 (delta 0), pack-reused 0
Unpacking objects: 100% (2/2), done.
From https://github.com/sj-science/WinterSchool_train_repo
 * [new branch] sj_branch -> origin/sj_branch
```

git branch -r

```
origin/HEAD -> origin/main
origin/main
origin/sj_branch
```

`git checkout sj_branch`

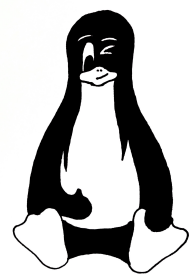
```
sj_branch
Switched to branch 'sj_branch'
Your branch is up to date with 'origin/sj_branch'.
```



* Create and switch to your own branch "your_initials_branch"
* Create another bash script with a message to your colleagues.
* Commit this file to the new branch
* Push the branch to track its remote counterpart
* Checkout one of your colleagues new branches and find out their message to you.

- **git** follows a distribution model where users maintain a local copy
  its main strength is its easy branching and merging capability
- hosting providers for git repositories **GitHub, GitLab, azure, Bitbucket** and others
- **svn** is a git alternative that follows a centralized model
- using git as a simple backup tool is slightly overkill considering its powerful features
- **onedrive** is a commercial cloud service (1TB included in VUW and NIWA accounts)
  - automatically creates an instant *commit+push* everytime a document is saved locally
  - version history can be accessed via Office-365
  - native client in MS-Win, clients available for Android, MacOS and Linux
- **seafile** is an opensource file sync&share engine (`www.seafile.com`)
  - comparable to onedrive seafile syncs instantly any changes with an online repository
  - works with command line & gui across all plattforms
  - SGEES offers 20GB server space through using seafile (talk to Aleks Beliaev)

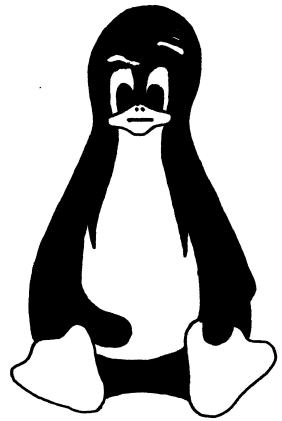
- **alternatives**
  - **Google Drive**, **MEGAsync**, **Amazon Cloud Drive**
  - many email providers offer a cloud service
  - the trick is to find a client app that works for your OS
  - **rsync** & external harddrive
  - **overleaf** for tex documents



∗ How about naming files in a suitable fashion?

6 | File Naming

- Establishing a good naming convention
  - helps to maintain structure inside folders
  - is the simplest method of version management & control
  - avoids duplicities
  - helps manually syncing folders
  - helps avoiding long names



- A good file name file_name.apx
  - contains all valid information to clearly identify the file content
  - should be no longer than 12 letters
  - **does NOT contain spaces**
  - leaves the file extension as is

- Examples
  023_bry_ACOSM_9km24_V02.nc
  article_draft_v12_sj1.docx
  RSSM_84Bg2bT3_4avg_abs_2118_fig1_Dec_Jan_966.png

6 | Data Handling - Access

- **Storing and access of large data files**
- Load only what you need (crop and slicing of data).
- Make sure filesystems are mounted sufficiently "close" to the processing instance.
  - Loading 1GB from a mounted network drive can test your patience
  - Sufficient USB3 link is now standard for portable drives
- Portable and therefore local drives are fine until processing becomes too demanding for the labtop
- Avoid duplicate copies
- Establish a universal mount point for data storage in your folder structure
  - avoids the need for hard coding (changing) access paths in processing scripts

6 | Data Handling - Backup

- **Backup strategy for data and model outputs**
  Depends on size and cost of reproducing.
- Collected Field data
  - Virtually not reproducible, unless you get another Marsden to revisit your field site
  - Needs fail save backup e.g. redundancy. Lab repository + Cloud Service + external disk.
- Model outputs from your own simulations.
  - Too big for redundant backups.
  - In case of disaster they can be reproduced by rerunning the model although there is a cost (only if a complete backup of the runtime script and input conditions exists!)
- Downloaded Data
  - If the repository (like CMIP database) is unlikely to be decommissioned no need for backup.

6 | Best Practise

> Know when to work quick-and-dirty or flash-and-orderly.
> Use scripts for anything > 3 lines.
> Save workflow for each of your products (figure, model output) under a unique name.
> Adopt a good naming convention. No spaces!
> Stay within your OS! Editing code or parsing lists in MS-Win and run under *nix can be disastrous.
> Separate IP from data in folder structure.
> Backup & version control all IP.
> Don't mix different cloud & versioning systems.
> Use cloud services to maintain synchronicity between individual work environments (desktops).
> Learn the basics of command line controls! :)