Boring but important disclaimers:

- If you are not getting this from the GitHub repository or the associated Canvas page (e.g. CourseHero, Chegg etc.), you are probably getting the substandard version of these slides Don't pay money for those, because you can get the most updated version for free at

  `https://github.com/julianmak/academic-notes`

  The repository principally contains the compiled products rather than the source for size reasons.
- Associated Python code (as Jupyter notebooks mostly) will be held on the same repository. The source data however might be big, so I am going to be naughty and possibly just refer you to where you might get the data if that is the case (e.g. JRA-55 data). I know I should make properly reproducible binders etc., but I didn't...
- I do not claim the compiled products and/or code are completely mistake free (e.g. I know I don't write Pythonic code). Use the material however you like, but use it at your own risk.
- As said on the repository, I have tried to honestly use content that is self made, open source or explicitly open for fair use, and citations should be there. If however you are the copyright holder and you want the material taken down, please flag up the issue accordingly and I will happily try and swap out the relevant material.

OCES 3301 :

basic Data Analysis in ocean sciences

Session 9: fun with maps

Outline

(Just overview here; for actual content see Jupyter notebooks)

- multi-dimensional arrays

  → documenting data varying in **space** (and in **time**)

  → basics of plotting (contours, animations) and statistics
- GEBCO and WOA13 data

  → NetCDF and `xarray`, basic functionalities

  → vertical profiles, meridional sections

  → masking, basic statistics, Hovmöller plots

Recap: data so far

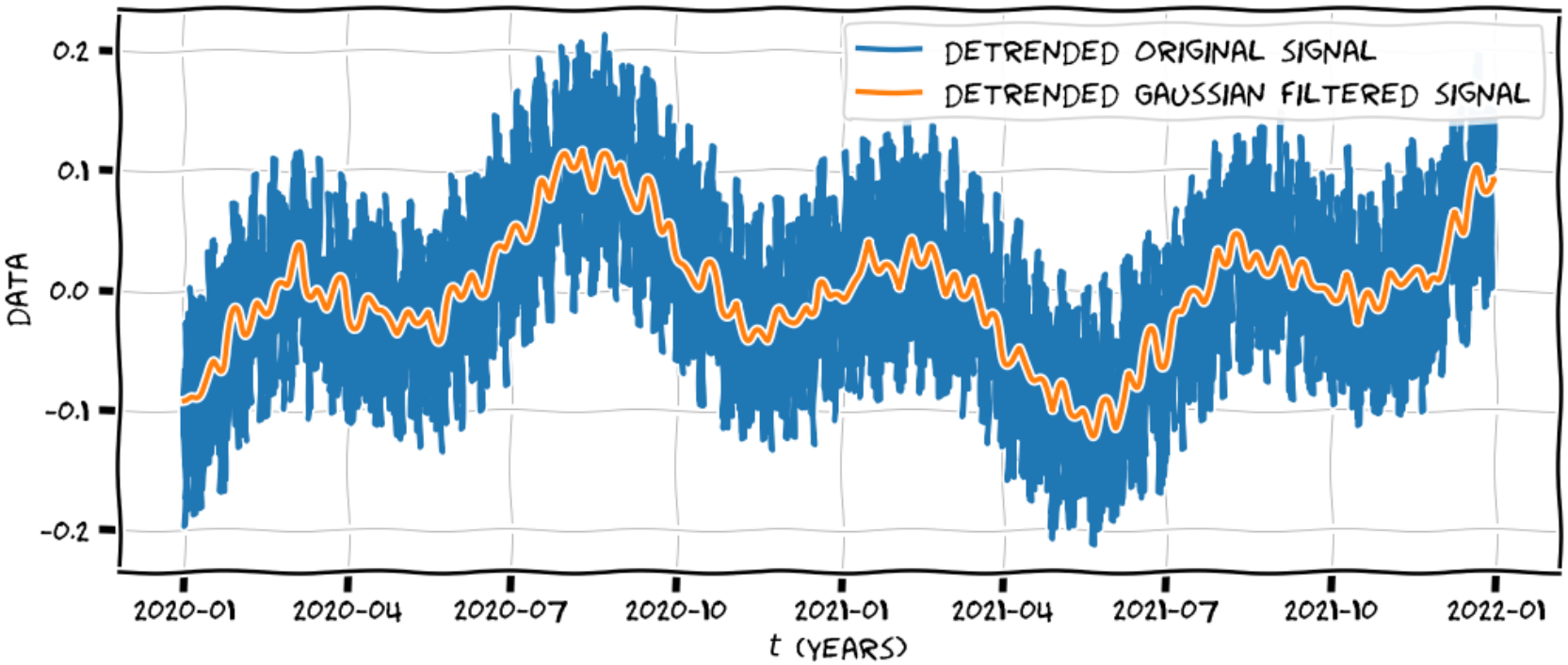


Figure: A detrended signal, as a 1d array varying in time.

Recap: data so far

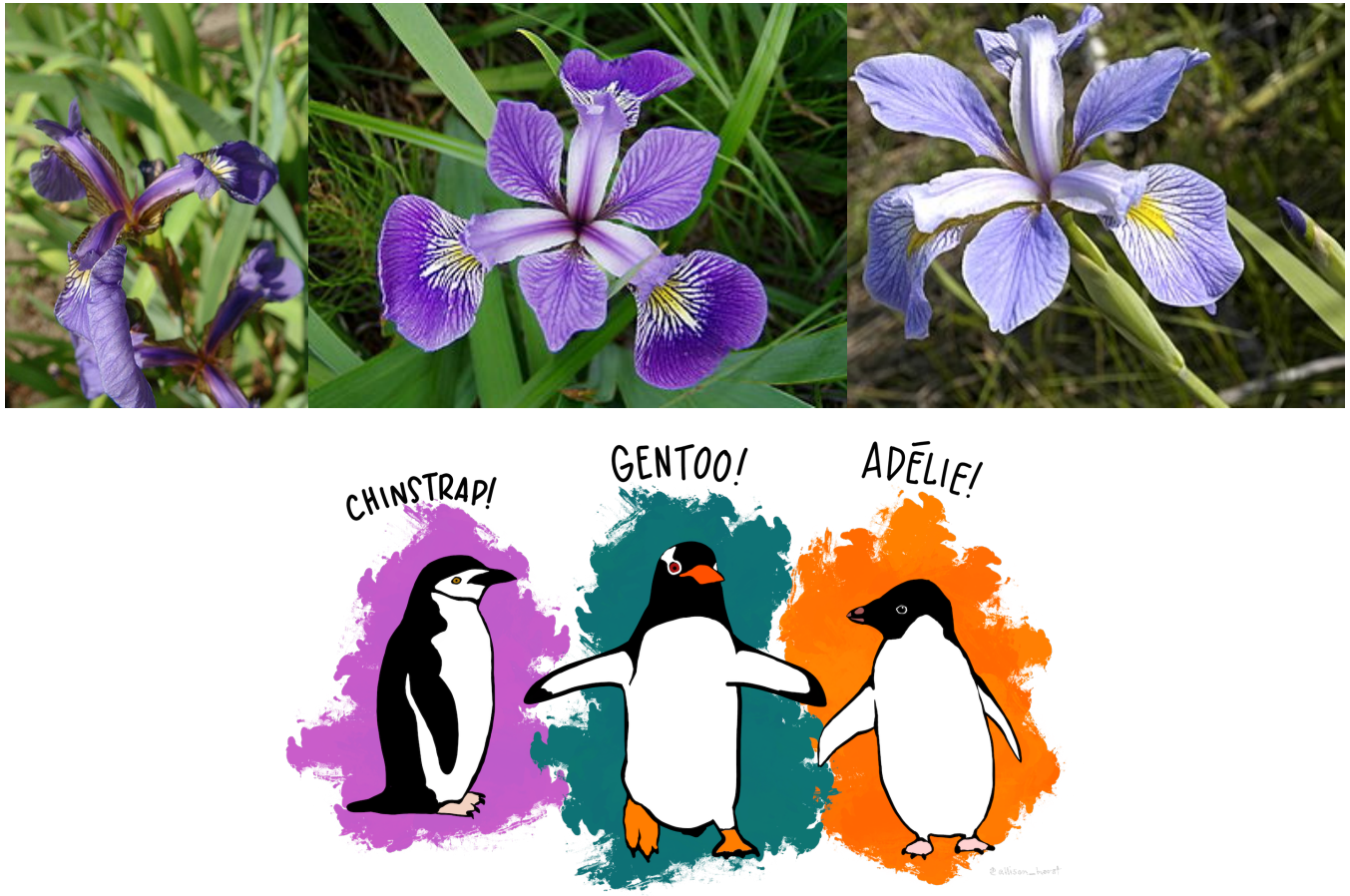


Figure: Analysis of data as 1d arrays against each other, but stored as multi-dimensional arrays.

Basic 2d array

Consider the function

$$f(x, y) = \sin(x)\cos(y)$$

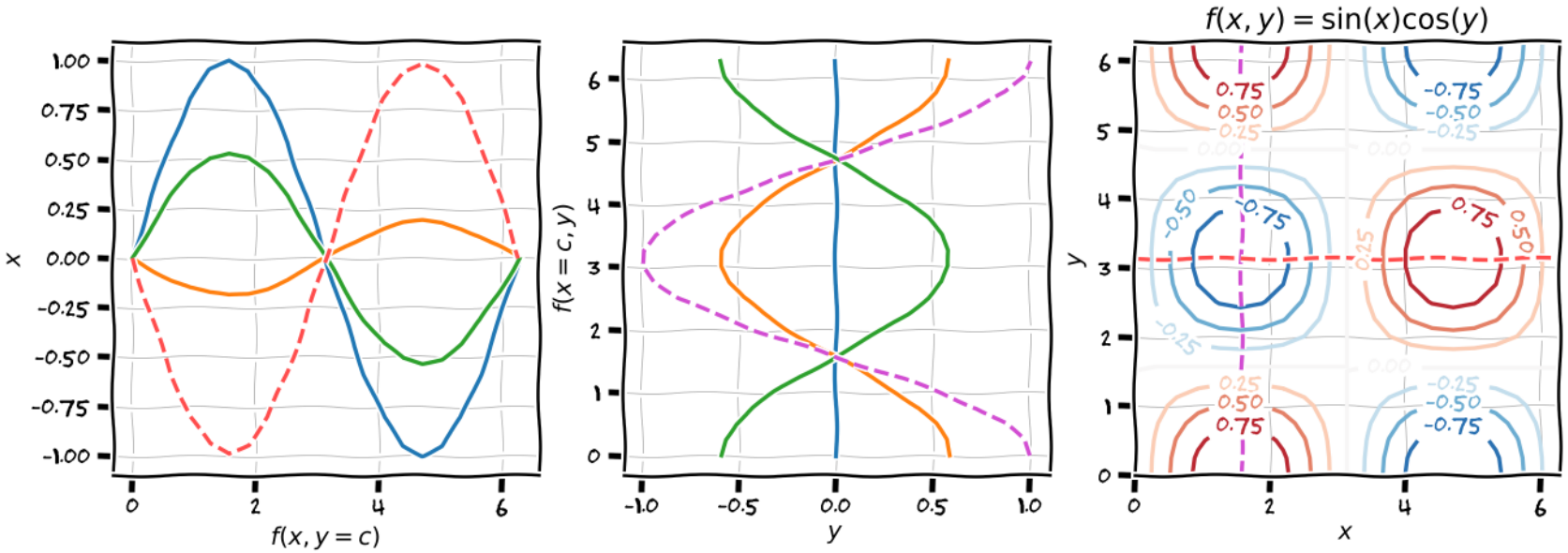


Figure: $f(x, y) = \sin(x)\cos(y)$ as lines at fixed y, fixed x, and contours in the (x, y).

- varying as sines in x and cosines in y
- circular 'blobs' in top-down view

Basic 2d array

$$f(x, y) = \sin(x)\cos(y)$$

Take

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N_x} \end{pmatrix}, \qquad y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N_y} \end{pmatrix}$$

and store as (see notebook to see how this would be generated)

$$f(x, y) = \begin{pmatrix} f(x_0, y_0) & f(x_1, y_0) & \cdots & f(x_{N_x}, y_0) \\ f(x_0, y_1) & f(x_1, y_1) & \cdots & f(x_{N_x}, y_1) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_0, y_{N_y}) & f(x_1, y_{N_y}) & \cdots & f(x_{N_x}, y_{N_y}) \end{pmatrix}.$$

- Python plotting 'prefers' storage of arrays in 'descending' dimensions `f[t,z,y,x]` etc.

Basic 2d array

$$f(x, y) = \sin(x)\cos(y)$$

- view the contour (or filled contour) plots as 'top-down' views of 3d plot

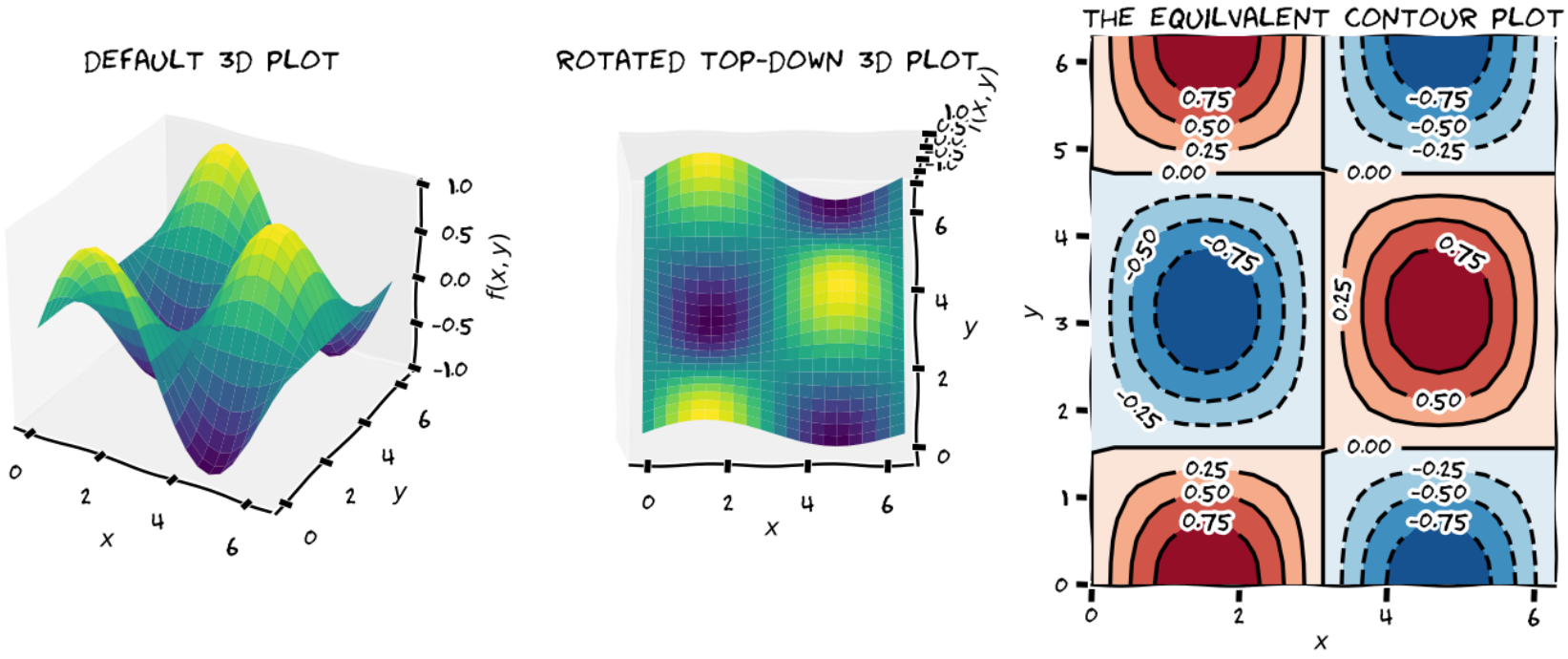


Figure: $f(x, y) = \sin(x)\cos(y)$ as 3d plots.

e.g. equation of state

- oceanography relevant 2d example: equation of state (EOS)

  $\rightarrow \rho = \rho(T, S, \cdots)$

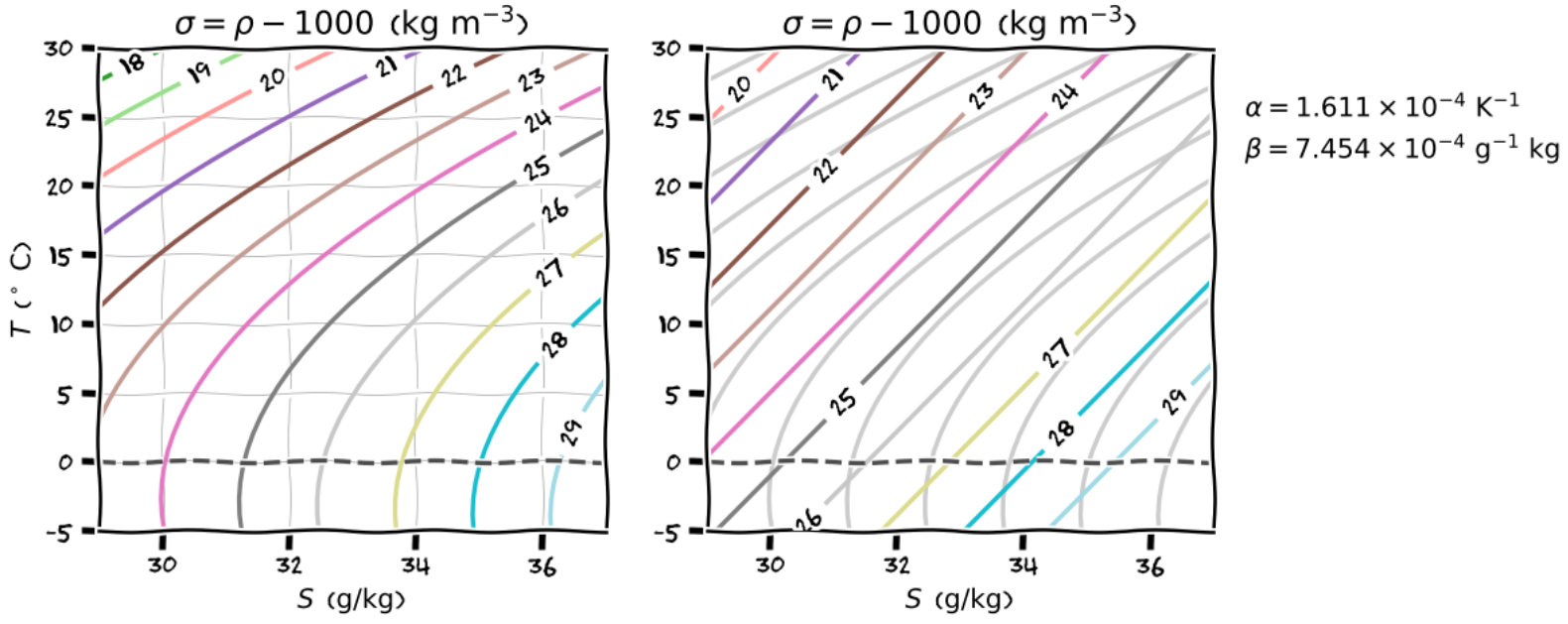


Figure: $\rho = \rho(T, S, \cdots)$ over choices of T and S, for the nonlinear TEOS-10 and linear EOS for some choice of α and β.

Animations

- really just pictures transitioning quickly

  $\rightarrow$ think flipbooks

  $\rightarrow$ a few still images as frames

  $\rightarrow$ transitioning with a frame rate (fps = frames per second)

(Humans perceive anything larger than about 16 fps as ‘smooth’, loosely speaking)

cursed beast most foul

(2 + 1)d array

$$f(x, y) = \sin(x)\cos(y)\sin(t)$$

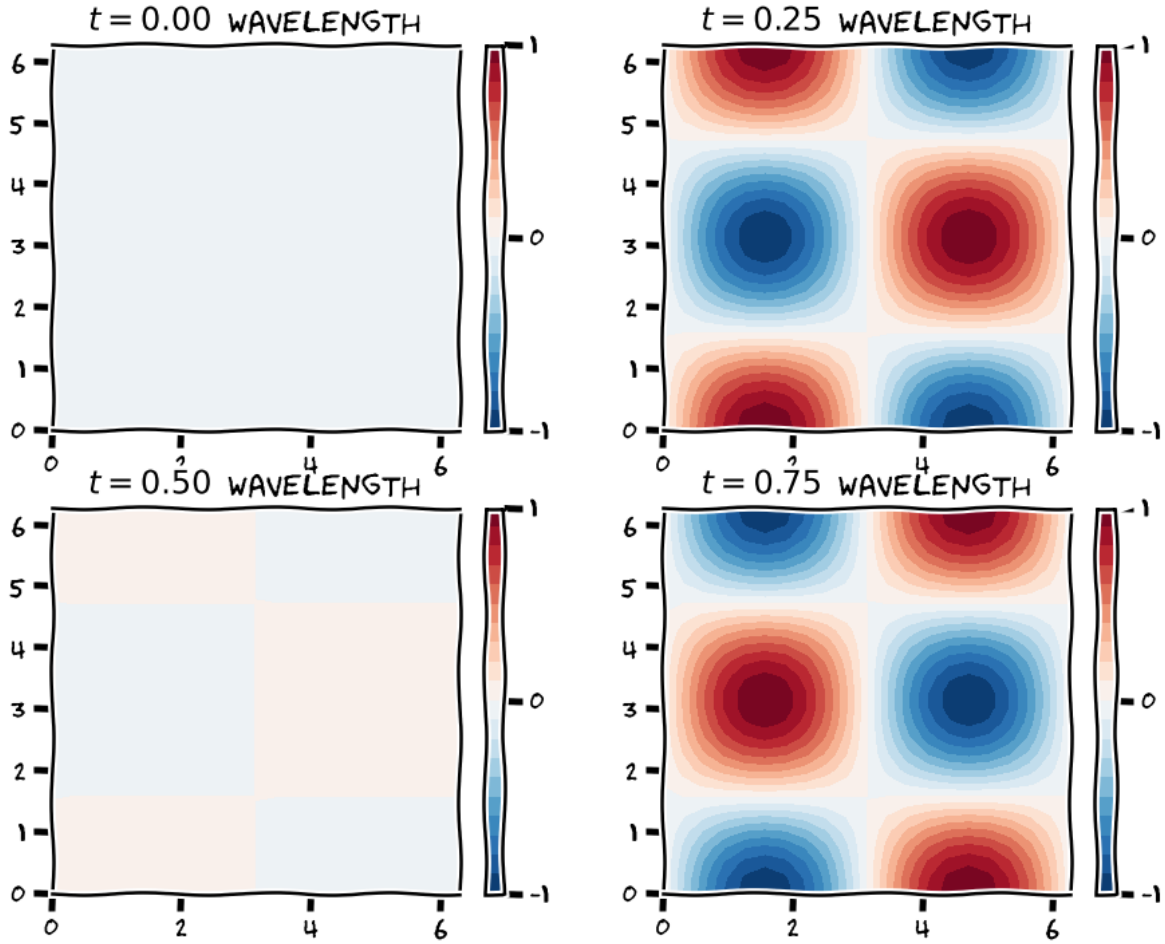


Figure: $f(x, y, t) = \sin(x)\cos(y)\sin(t)$ at some fixed times.

(2 + 1)d array

$$f(x, y) = \sin(x) \cos(y) \sin(t)$$

- data stored as `f[t_k, y_j, x_i]`
- see code for how you might animate the above in Python/Jupyter

(I tend to output all frames in Python but assemble into animation externally)

wibbly wobbly timey wimey

(2 + 1)d array: statistics

$$f(x, y) = \sin(x)\cos(y)\sin(t)$$

- can compute averages, stds, rolling-averages etc.
  → over t (time average)
  → over space (spatial average)
  → over the longitude or x (zonal average)

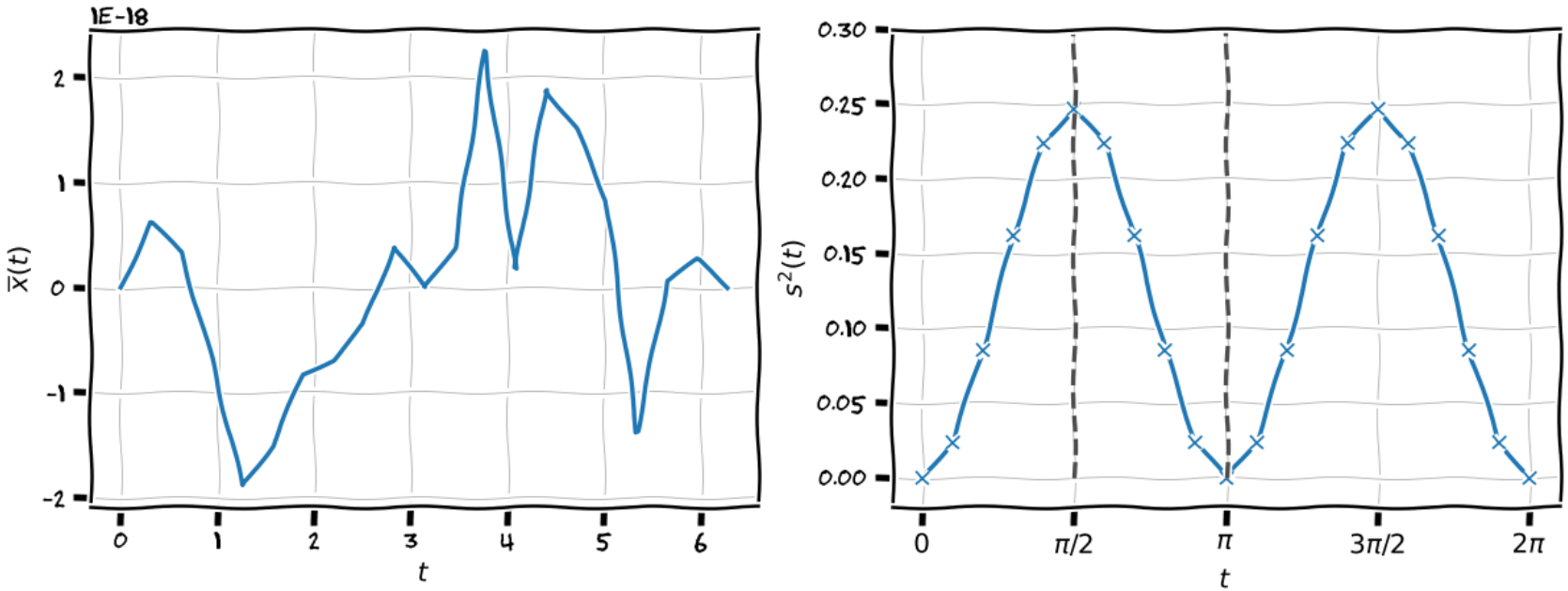


Figure: Spatial average and variance of $f(x, y, t) = \sin(x)\cos(y)\sin(t)$ as a function of time.

GEBCO and WOA13 data

More 'realistic' examples:

- GEBCO (The General Bathymetric Chart of the Oceans)

  → topography (above sea level) and bathymetry (below sea level) I will probably use them interchangeably

  → various spatial resolutions, time independent

  → $\text{elev}(\text{lat}, \text{lon})$

- WOA13 (World Ocean Atlas 2013)

  → various quantities, going to deal with T and S here

  → climatologies, monthly ('average' monthly condition)

  → 1° spatial resolution (but will find denser interpolated versions)

  → $f(\text{month}, \text{depth}, \text{lat}, \text{lon})$

GEBCO and WOA13 data

Geophysical data normally comes as NetCDF (Network Common Data Form)

- with extension `*.nc` or sometimes `*.nc4`

  → self-describing, normally includes co-ordinates files, masks (see later), attributes etc.

  → need a library to read it

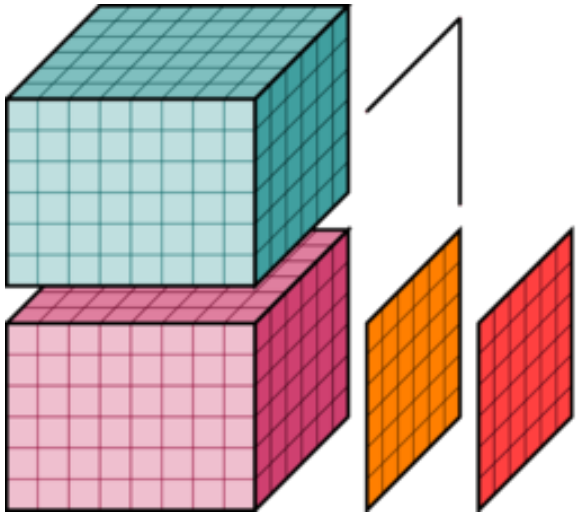
NetCDF and CF: The Basics

GEBCO and WOA13 data

- going to use `xarray` to read NetCDF files

  → could think of it as a convenient front end for the Python NetCDF library

  → actually much more powerful than that (see notebook)



xarray

GEBCO and WOA13 data

- xarray returns a data frame **`df`**

  → cf. Pandas data frame

  → see co-ordinate variables, data variables, attributes etc.

```
<xarray.Dataset>
Dimensions:  (lat: 451, lon: 901)
Coordinates:
  * lat      (lat) float64 -90.0 -89.6 -89.2 -88.8 -88.4 ... 88.8 89.2 89.6 90.0
  * lon      (lon) float64 -180.0 -179.6 -179.2 -178.8 ... 179.2 179.6 180.0
Data variables:
    elev     (lat, lon) float32 ...
Attributes:
    Conventions:  CF-1.6
    title:        The GEBCO One Minute Grid - a continuous terrain model for ...
    institution:  On behalf of the General Bathymetric Chart of the Oceans (G...
    source:       The grid is largely based on the bathymetric contours conta...
    history:      This is version 2.0 of the data set, released in November 2008
    references:   Information on the data set is available from the internet:...
    comment:      The data in the GEBCO One Minute Grid should not be used fo...
    node_offset:  0
```

Figure: Sample output from GEBCO read via xarray.

GEBCO and WOA13 data: surface plots

- xarray objects have imbued functions

  → `df["elev"].plot()` below with GEBCO data

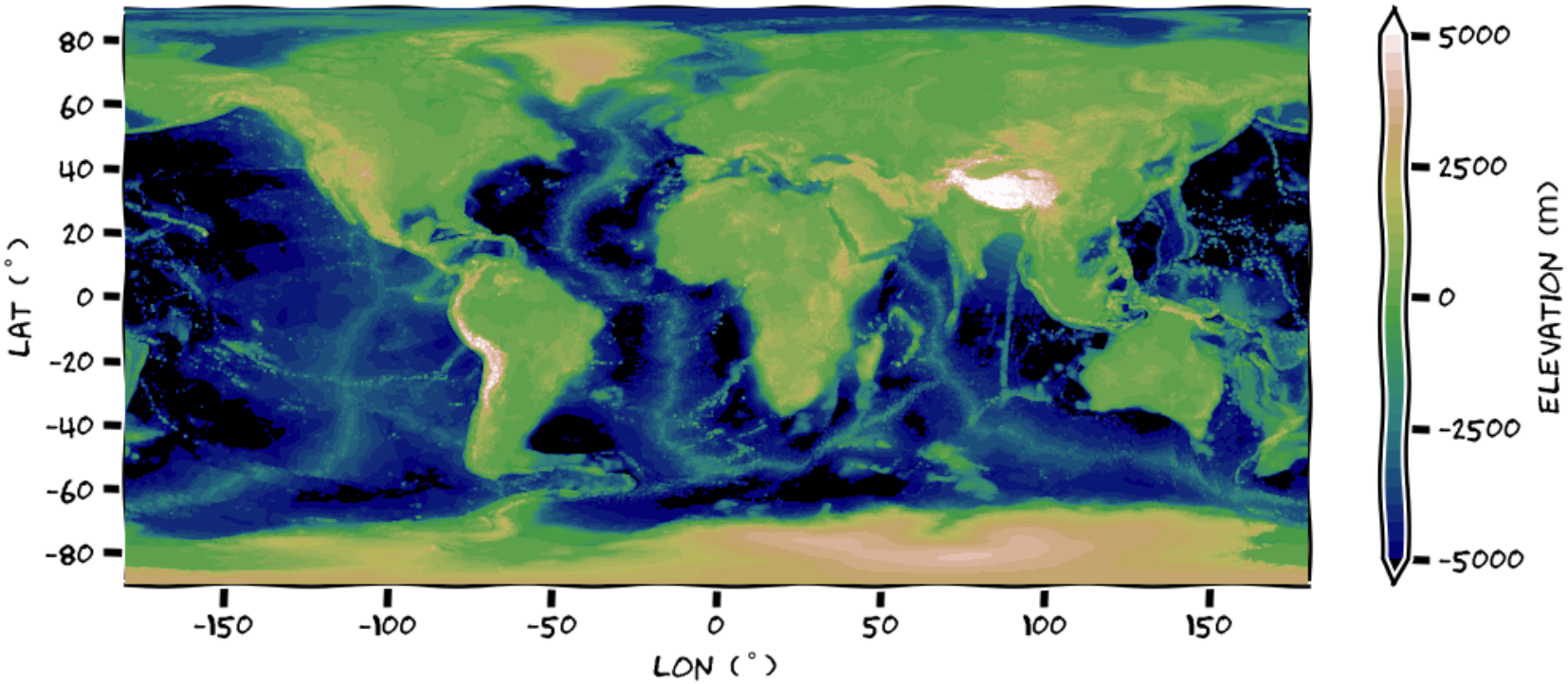


Figure: Sample output from GEBCO read via xarray.

GEBCO and WOA13 data: vertical profile

- xarray objects have imbued functions
  → `df["votemper"].sel().plot()` to pick out lon/lat/time locations rather than specifying the index explicitly, and doing plot

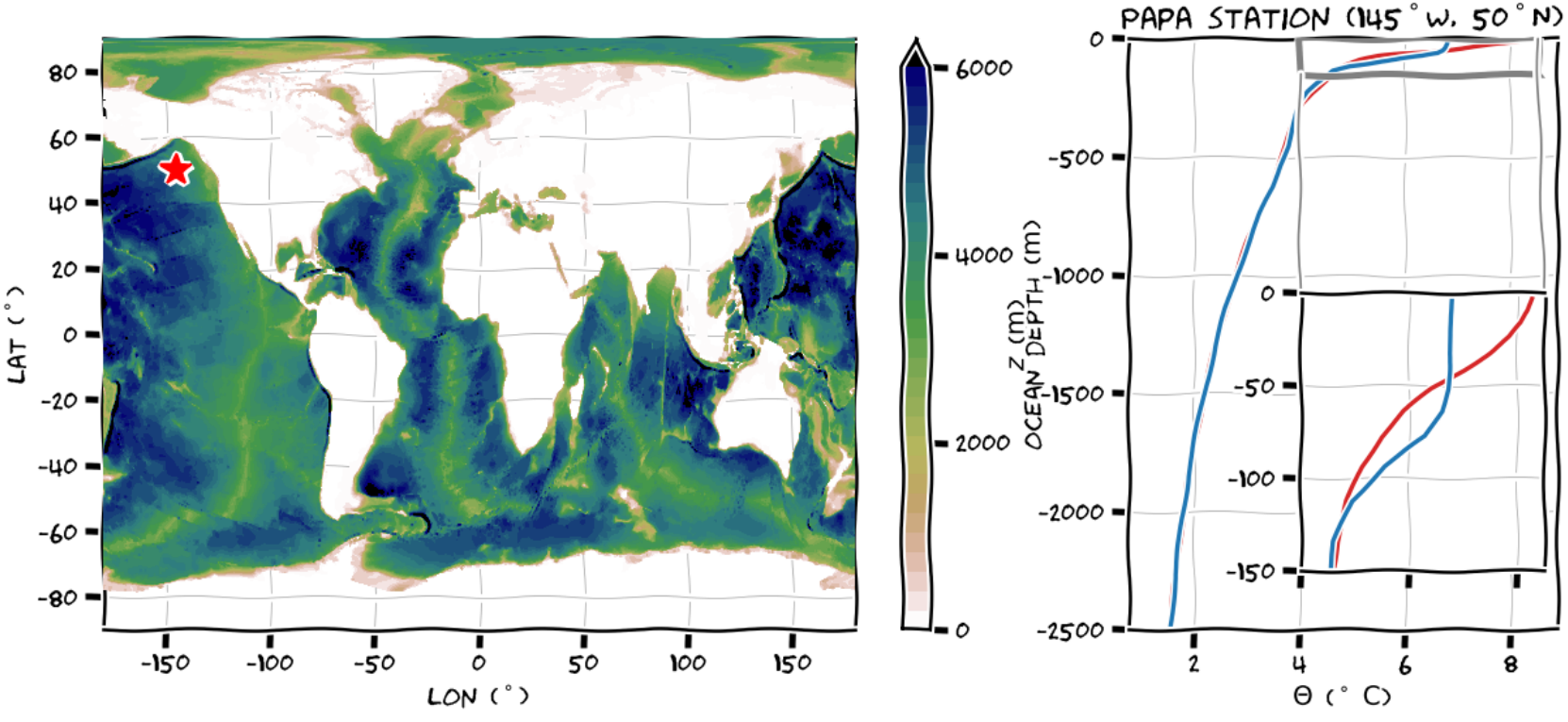


Figure: Vertical profile of WOA13 (conservative) temperature at summer and winter months at PAPA station.

GEBCO and WOA13 data: meridional section

- as previous, but for meridional section
  → fixed latitude, function of lon and depth
  → `.sel(lat=SOMETHING, method="nearest")`

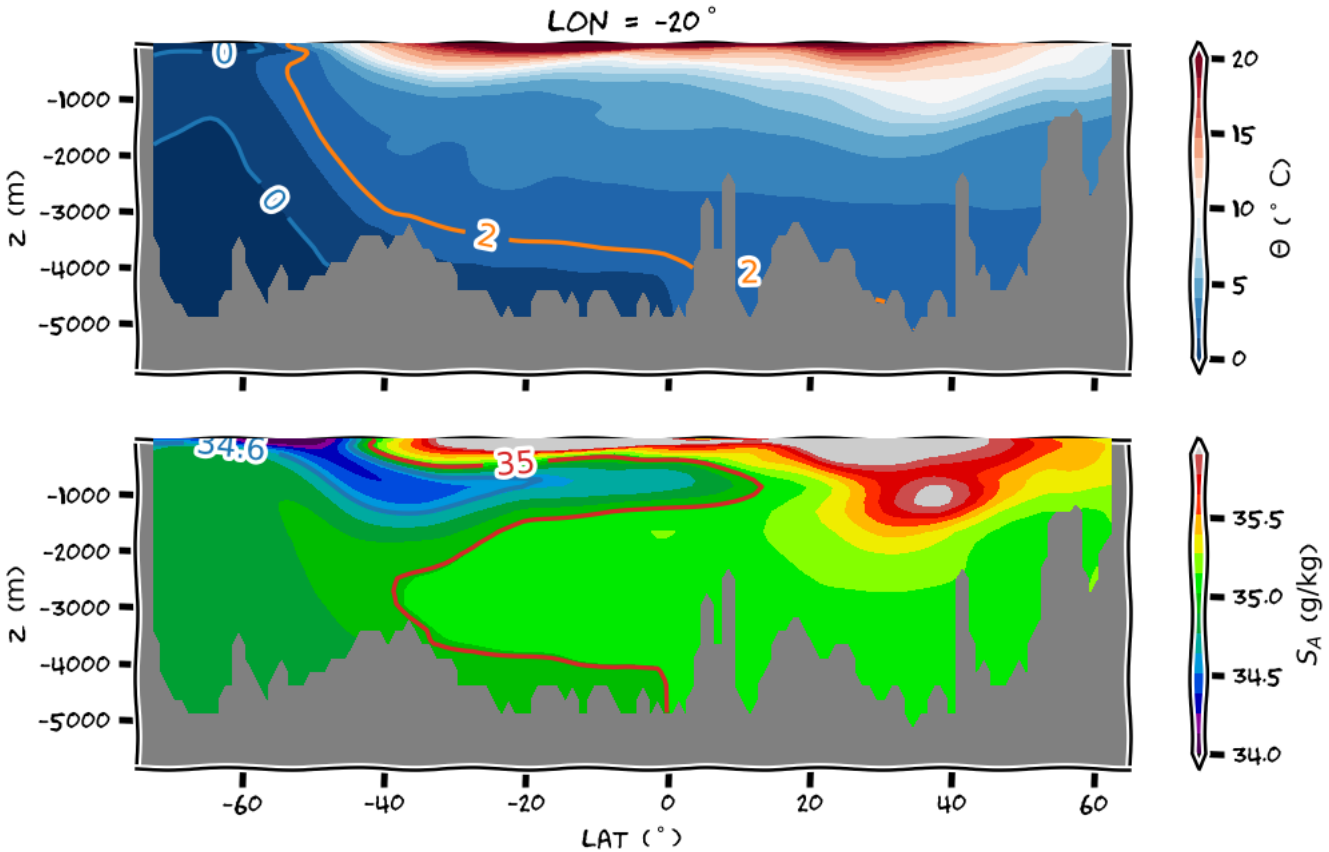


Figure: Meridional profile of time-averaged WOA13 (conservative) temperature and (absolute) salinity. Masking of based on GEBCO data.

GEBCO and WOA13 data: masking

- WOA13 data here is not masked

  → e.g. $f(\text{month}, \text{depth}, \text{lat}, \text{lon})$ contains co-ordinates at land points

  → wanted to exclude those points when doing calculations (e.g. averages)

- create a mask separately here from GEBCO data

  → boolean arrays of `True/False` or integer arrays of `1/0` to correspond to ocean/land points

  → using some conditional, e.g. `if df["elev"] < 0 then 1 else 0`

GEBCO and WOA13 data: masking

- mask is (`df["elev"] < 0`) AND (lon and lat within some region)
  $\rightarrow$ 1 if true, 0 otherwise
- construct $T \times$ mask
  $\rightarrow$ I actually used NaNs instead of 0

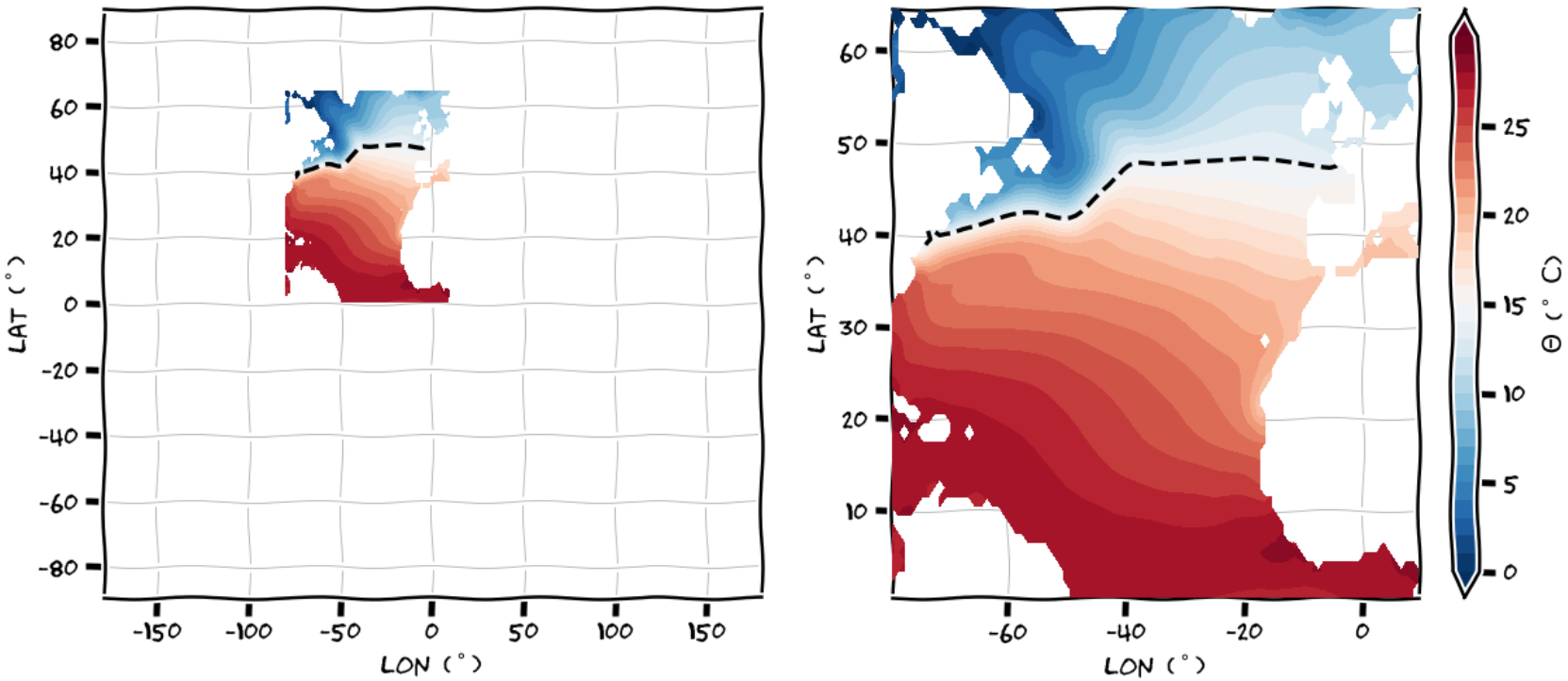


Figure: Mask created GEBCO data based on land, and within a certain region in the Atlantic. Plotting the surface WOA13 temperature on top of that.

GEBCO and WOA13 data: masking and stats

NaNs instead of zeroes might be preferably because

- plotting commands normally ignore NaNs (see above)

  $\rightarrow$ e.g. if zeroes, then contours of '0' will probably dominate

- there are routines to ignore NaNs in calculations (e.g. `np.nanmean`)

  $\rightarrow$ e.g.

$$\begin{aligned} \text{nanmean}\{1,2,3,\text{NaN}\} &= \frac{1+2+3}{3} &&= 2 \\ \text{mean}\{1,2,3,0\} &= \frac{1+2+3+0}{4} &&= 1.5 \\ \text{mean}\{1,2,3,\text{NaN}\} &= \frac{1+2+3+\text{NaN}}{4} &&= ??? \end{aligned}$$

GEBCO and WOA13 data: masking and stats

Domain average (really just nanmean twice here) and zonal average (really just nanmean over longitude)

$$\langle \Theta \rangle = \frac{1}{A} \iint \Theta \, \mathrm{d}\boldsymbol{A} = \frac{1}{A} \iint \Theta \, \mathrm{d}x \, \mathrm{d}y \qquad \overline{\Theta} = \frac{1}{L_x} \int \Theta \, \mathrm{d}x.$$

- $\langle \Theta \rangle$ is a function of t only, while $\overline{\Theta}$ is a function of t and y

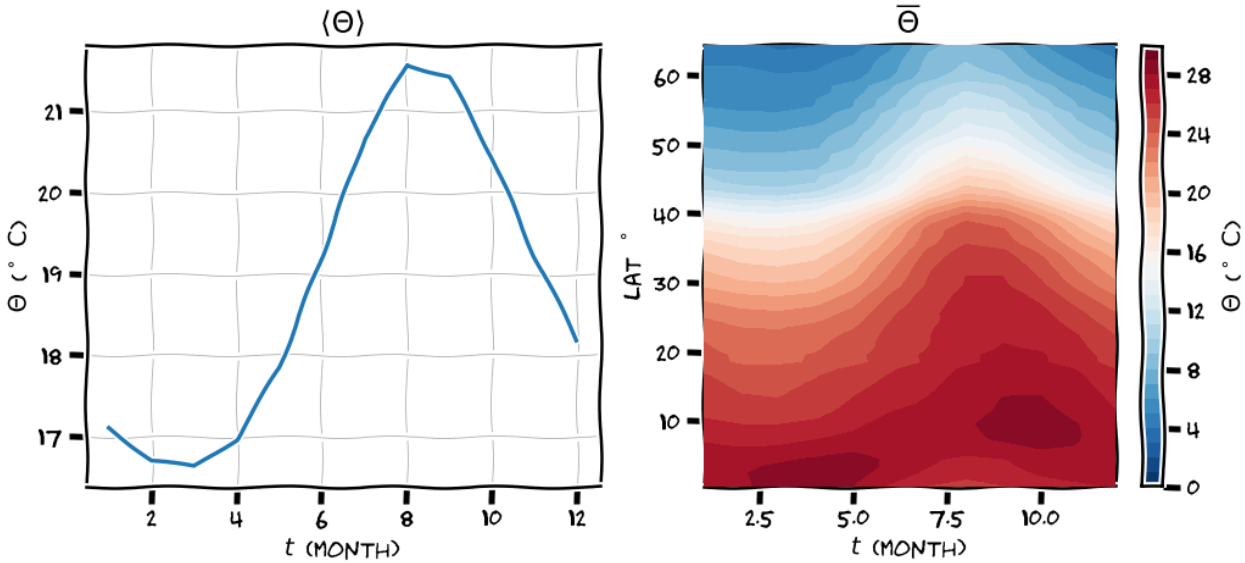


Figure: Domain and zonal average of WOA13 surface temperature over the Atlantic, as a time-series and a Hovmöller diagram.

Jupyter notebook

go to 09 Jupyter notebook to get some code practise

- try analyses on some sample data from a numerical simulation also

  → surface current speed from NEMO ORCA0083-N01

  → 1 year, every 5 days, nominally 1/12° horizontal resolution

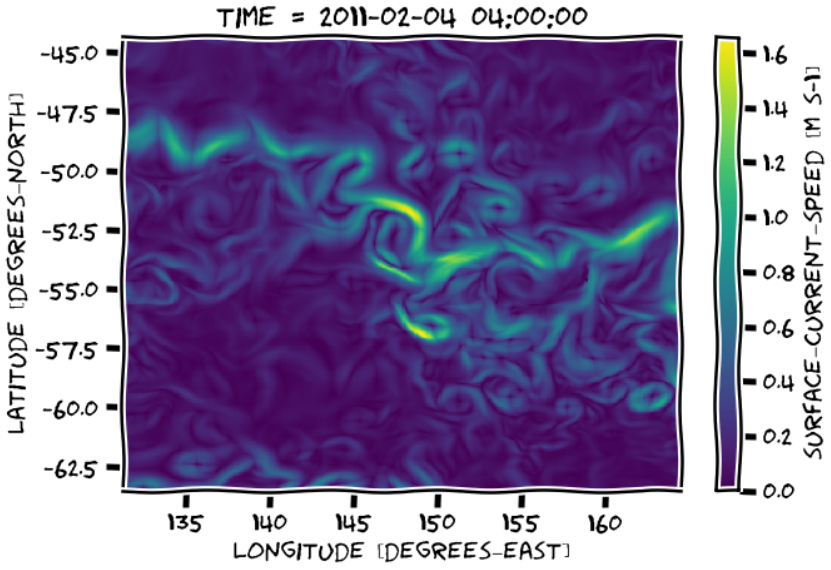


Figure: Surface current speed from NEMO ORCA0083-N01 somewhere in the Southern Ocean (ask me for more data if you want).