Introduction into working with ocean simulation output

The Team

Ryan Patmore

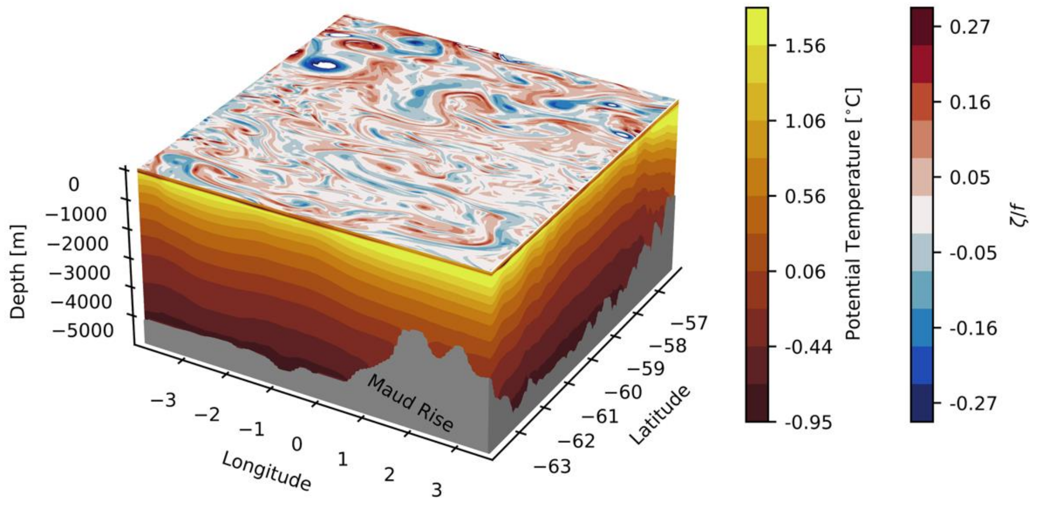
Julia Rulent

Jonathan Coney

Bablu Sinha

The Aims

Hands-on appreciation for the nature of model data



Patmore et al. (2024)

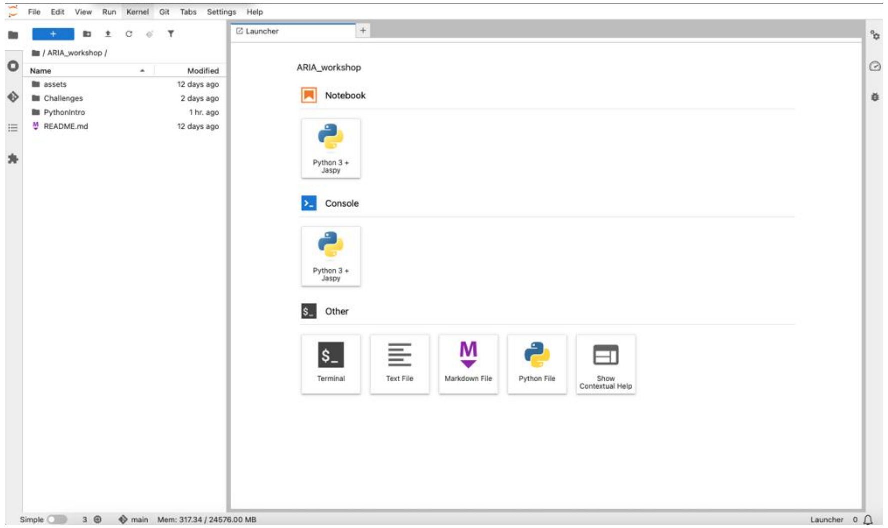
Analysis methods for identifying tipping points

The Platforms

- JASMIN Jupyter notebooks for exploring the data
- Split into teams of 4/5 to tackle 2+ challenges
- Challenges outlined in ARIA_workshop repo

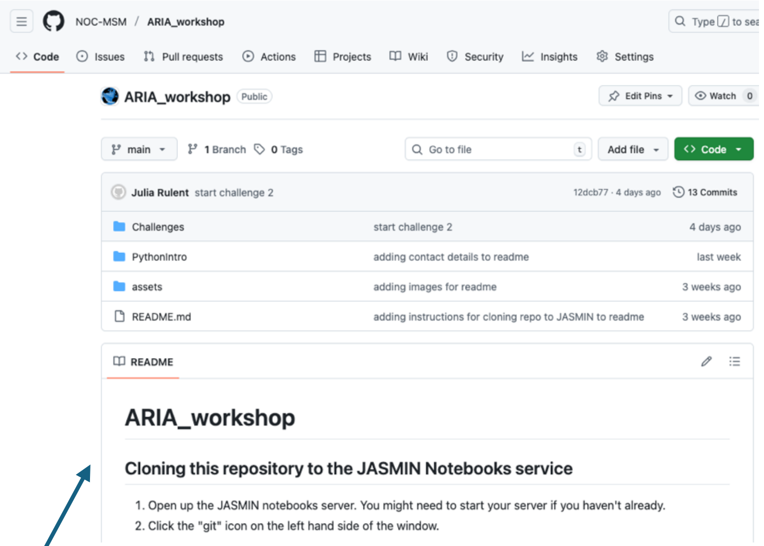
Notebook server

https://notebooks.jasmin.ac.uk/



ARIA_workshop Repository

https://github.com/NOC-MSM/ARIA_workshop



Instructions on landing page

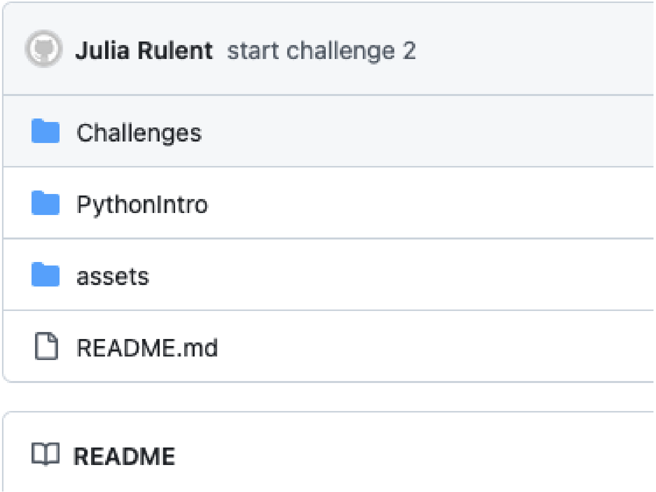
VERIFY

The Content

- Reference content for using Python
- See Jonathan for advice

ARIA_workshop Repository

Introductory Modules



PythonIntro introductory modules

- Thank you for filling in the poll with your Python ability.
- The Challenge notebooks assume knowledge of *some* Python, particularly familiarity with the libraries `xarray` and `matplotlib`.
- If you're fairly new to Python, unfamiliar with these libraries, or fancy refreshing your memory:
  - The two notebooks in the `PythonIntro/` directory are designed to be a whistle-stop tour of some basic Python (notebook 0) and an introduction to analysing and plotting data (notebook 1), in readiness for the Challenge notebooks.
  - Designed to be self-led, but if something doesn't make sense, **ask for help**!

It's all here:

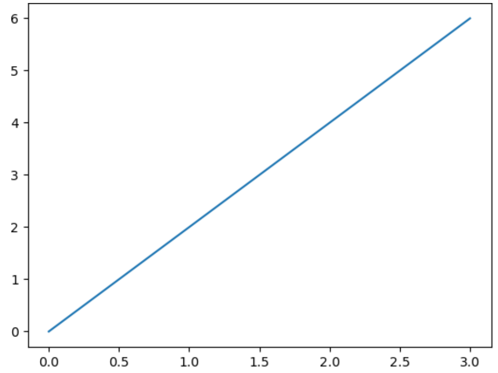
Print statements

```
print("Hello world!")
```

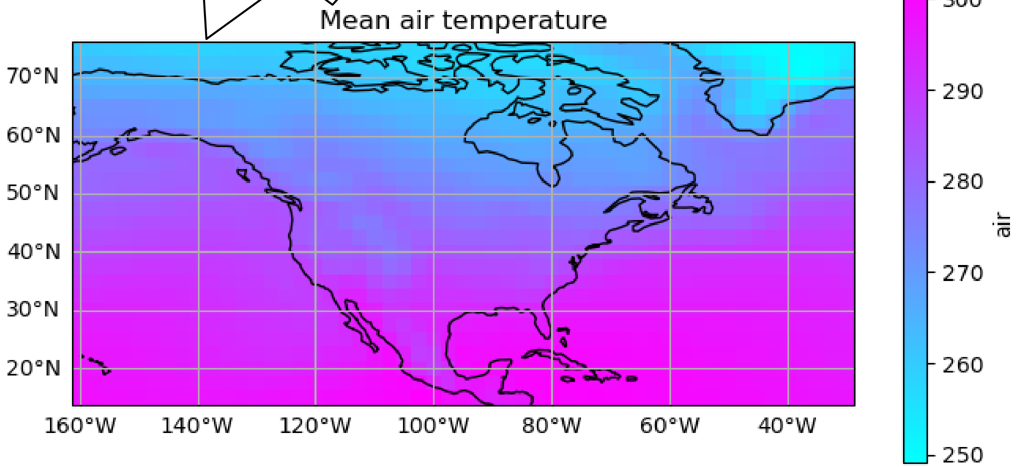
```
binomial_name = "Drosophila melanogaster"
group = binomial_name[0:10]
```

slicing

Plotting



And map projections? You bet!

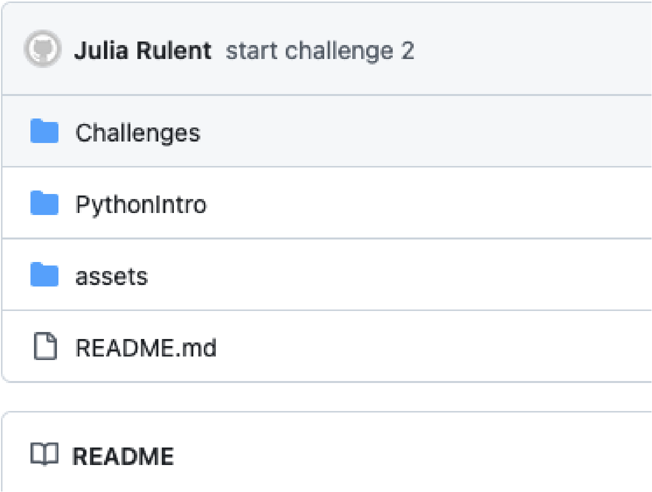


The Content

Main content sits in Challenges directory

ARIA_workshop Repository

The group challenges



VERIFY

The Content

Work through three challenges over 3x 1.5 hour sessions

ARIA_workshop Repository

Challenge 1

“Identify tipping points in the CLASS model data”

Name
..
Challenge1.ipynb
Challenge2.ipynb
Challenge3_advanced.ipynb
README.md

README.md

VERIFY

The Content

Work through three challenges over 3x 1.5 hour sessions

ARIA_workshop Repository

Challenge 2

“Demonstrate how one might devise an observational campaign to capture this tipping point”

Name
..
Challenge1.ipynb
Challenge2.ipynb
Challenge3_advanced.ipynb
README.md

README.md

VERIFY

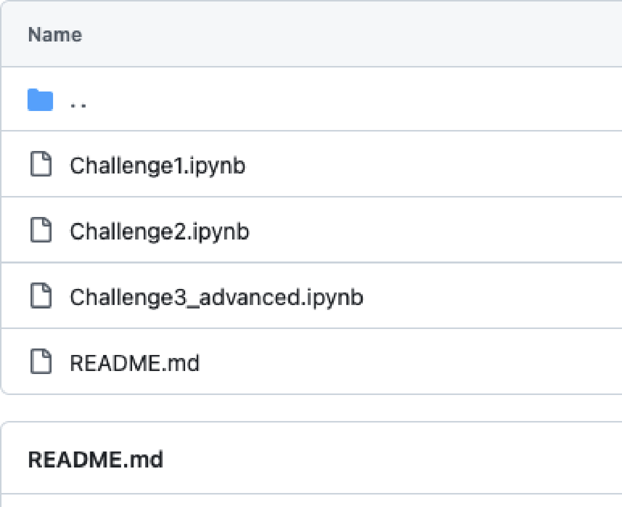
The Content

Work through three challenges over 3x 1.5 hour sessions

ARIA_workshop Repository

“Derive tipping point metrics”

Challenge 3 (Advanced)



VERIFY

The Content

Are we all set up with the JASMIN notebooks?

Challenge 1: Introduction

Grid Information

```
dom_cfg = xr.open_dataset(path + "domain_cfg.nc")
dom_cfg
```

```
Out[39]:
<xarray.Dataset> Size: 8GB
Dimensions:        (y: 1207, x: 1442, nav_lev: 75, time_counter: 1, nlines: 67)
Coordinates:
  * nav_lev        (nav_lev) float32 300B 0.5058 1.556 ... 5.698e+03 5.902e+03
  * time_counter   (time_counter) float64 8B 0.0
Dimensions without coordinates: y, x, nlines
Data variables: (12/43)
    nav_lon        (y, x) float32 7MB ...
    nav_lat        (y, x) float32 7MB ...
    jpiglo         int32 4B ...
    jpjglo         int32 4B ...
    jpkglo         int32 4B ...
    jperio         int32 4B ...
    ...             ...
    bottom_level   (time_counter, y, x) int32 7MB ...
    top_level      (time_counter, y, x) int32 7MB ...
    isf_draft      (time_counter, y, x) float64 14MB ...
    bathy_metry    (time_counter, y, x) float64 14MB ...
    namelist_cfg   (nlines) |S102 7kB ...
    closea_mask    (y, x) float64 14MB ...
Attributes:
    DOMAIN_number_total:     1
    DOMAIN_number:           0
    DOMAIN_dimensions_ids:   [1 2]
    DOMAIN_size_global:      [1442 1207]
    DOMAIN_size_local:       [1442 1207]
    DOMAIN_position_first:   [1 1]
    DOMAIN_position_last:    [1442 1207]
    DOMAIN_halo_size_start:  [0 0]
    DOMAIN_halo_size_end:    [0 0]
    DOMAIN_type:             BOX
```

Challenge 1: Introduction

Grid Information

Depth and time coordinates

```
dom_cfg = xr.open_dataset(path + "domain_cfg.nc")
dom_cfg
```

```
Out[39]:
<xarray.Dataset> Size: 8GB
Dimensions:        (y: 1207, x: 1442, nav_lev: 75, time_counter: 1, nlines: 67)
Coordinates:
  * nav_lev        (nav_lev) float32 300B 0.5058 1.556 ... 5.698e+03 5.902e+03
  * time_counter   (time_counter) float64 8B 0.0
Dimensions without coordinates: y, x, nlines
Data variables: (12/43)
    nav_lon        (y, x) float32 7MB ...
    nav_lat        (y, x) float32 7MB ...
    jpiglo         int32 4B ...
    jpjglo         int32 4B ...
    jpkglo         int32 4B ...
    jperio         int32 4B ...
    ...             ...
    bottom_level   (time_counter, y, x) int32 7MB ...
    top_level      (time_counter, y, x) int32 7MB ...
    isf_draft      (time_counter, y, x) float64 14MB ...
    bathy_metry    (time_counter, y, x) float64 14MB ...
    namelist_cfg   (nlines) |S102 7kB ...
    closea_mask    (y, x) float64 14MB ...
Attributes:
    DOMAIN_number_total:     1
    DOMAIN_number:           0
    DOMAIN_dimensions_ids:   [1 2]
    DOMAIN_size_global:      [1442 1207]
    DOMAIN_size_local:       [1442 1207]
    DOMAIN_position_first:   [1 1]
    DOMAIN_position_last:    [1442 1207]
    DOMAIN_halo_size_start:  [0 0]
    DOMAIN_halo_size_end:    [0 0]
    DOMAIN_type:             BOX
```

VERIFY

Challenge 1: Introduction

Grid Information

Latitude and Longitude

```
dom_cfg = xr.open_dataset(path + "domain_cfg.nc")
dom_cfg
```

```
Out[39]: <xarray.Dataset> Size: 8GB
Dimensions:          (y: 1207, x: 1442, nav_lev: 75, time_counter: 1, nlines: 67)
Coordinates:
  * nav_lev          (nav_lev) float32 300B 0.5058 1.556 ... 5.698e+03 5.902e+03
  * time_counter     (time_counter) float64 8B 0.0
Dimensions without coordinates: y, x, nlines
Data variables: (12/43)
    nav_lon          (y, x) float32 7MB ...
    nav_lat          (y, x) float32 7MB ...
    jpiglo           int32 4B ...
    jpjglo           int32 4B ...
    jpkglo           int32 4B ...
    jperio           int32 4B ...
    ...               ...
    bottom_level     (time_counter, y, x) int32 7MB ...
    top_level        (time_counter, y, x) int32 7MB ...
    isf_draft        (time_counter, y, x) float64 14MB ...
    bathy_metry      (time_counter, y, x) float64 14MB ...
    namelist_cfg     (nlines) |S102 7kB ...
    closea_mask      (y, x) float64 14MB ...
Attributes:
    DOMAIN_number_total:      1
    DOMAIN_number:            0
    DOMAIN_dimensions_ids:    [1 2]
    DOMAIN_size_global:       [1442 1207]
    DOMAIN_size_local:        [1442 1207]
    DOMAIN_position_first:    [1 1]
    DOMAIN_position_last:     [1442 1207]
    DOMAIN_halo_size_start:   [0 0]
    DOMAIN_halo_size_end:     [0 0]
    DOMAIN_type:              BOX
```

VERIFY

Challenge 1: Introduction

Grid Information

The remainder is general grid information

```
dom_cfg = xr.open_dataset(path + "domain_cfg.nc")
dom_cfg
```

```
Out[39]: <xarray.Dataset> Size: 8GB
Dimensions:        (y: 1207, x: 1442, nav_lev: 75, time_counter: 1, nlines: 67)
Coordinates:
  * nav_lev        (nav_lev) float32 300B 0.5058 1.556 ... 5.698e+03 5.902e+03
  * time_counter   (time_counter) float64 8B 0.0
Dimensions without coordinates: y, x, nlines
Data variables: (12/43)
    nav_lon        (y, x) float32 7MB ...
    nav_lat        (y, x) float32 7MB ...
    jpiglo         int32 4B ...
    jpjglo         int32 4B ...
    jpkglo         int32 4B ...
    jperio         int32 4B ...
    ...             ...
    bottom_level   (time_counter, y, x) int32 7MB ...
    top_level      (time_counter, y, x) int32 7MB ...
    isf_draft      (time_counter, y, x) float64 14MB ...
    bathy_metry    (time_counter, y, x) float64 14MB ...
    namelist_cfg   (nlines) |S102 7kB ...
    closea_mask    (y, x) float64 14MB ...
Attributes:
    DOMAIN_number_total:      1
    DOMAIN_number:            0
    DOMAIN_dimensions_ids:    [1 2]
    DOMAIN_size_global:       [1442 1207]
    DOMAIN_size_local:        [1442 1207]
    DOMAIN_position_first:    [1 1]
    DOMAIN_position_last:     [1442 1207]
    DOMAIN_halo_size_start:   [0 0]
    DOMAIN_halo_size_end:     [0 0]
    DOMAIN_type:              BOX
```

VERIFY

Challenge 1: Introduction

Grid Information

```
dom_cfg = xr.open_dataset(path + "domain_cfg.nc")
dom_cfg
```

```
Out[39]: <xarray.Dataset> Size: 8GB
Dimensions:        (y: 1207, x: 1442, nav_lev: 75, time_counter: 1, nlines: 67)
Coordinates:
  * nav_lev        (nav_lev) float32 300B 0.5058 1.556 ... 5.698e+03 5.902e+03
  * time_counter   (time_counter) float64 8B 0.0
Dimensions without coordinates: y, x, nlines
Data variables: (12/43)
    nav_lon        (y, x) float32 7MB ...
    nav_lat        (y, x) float32 7MB ...
    jpiglo         int32 4B ...
    jpjglo         int32 4B ...
    jpkglo         int32 4B ...
    jperio         int32 4B ...
    ...             ...
    bottom_level   (time_counter, y, x) int32 7MB ...
    top_level      (time_counter, y, x) int32 7MB ...
    isf_draft      (time_counter, y, x) float64 14MB ...
    bathy_metry    (time_counter, y, x) float64 14MB ...
    namelist_cfg   (nlines) |S102 7kB ...
    closea_mask    (y, x) float64 14MB ...
Attributes:
    DOMAIN_number_total:     1
    DOMAIN_number:           0
    DOMAIN_dimensions_ids:   [1 2]
    DOMAIN_size_global:      [1442 1207]
    DOMAIN_size_local:       [1442 1207]
    DOMAIN_position_first:   [1 1]
    DOMAIN_position_last:    [1442 1207]
    DOMAIN_halo_size_start:  [0 0]
    DOMAIN_halo_size_end:    [0 0]
    DOMAIN_type:             BOX
```

The remainder is general grid information

Let's delve further...

VERIFY

Challenge 1: Introduction

NEMO uses the commonly adopted Arakawa C-Grid (Arakawa and Lamb, 1977)

Challenge 1: Introduction

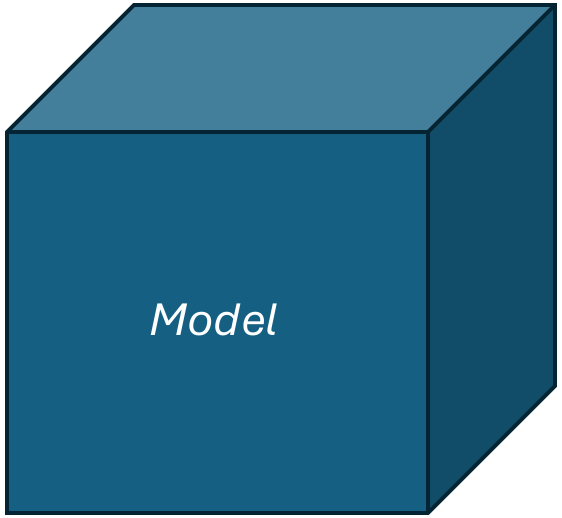
NEMO uses the commonly adopted Arakawa C-Grid (Arakawa and Lamb, 1977)

It also runs on Fortran!

Challenge 1: Introduction

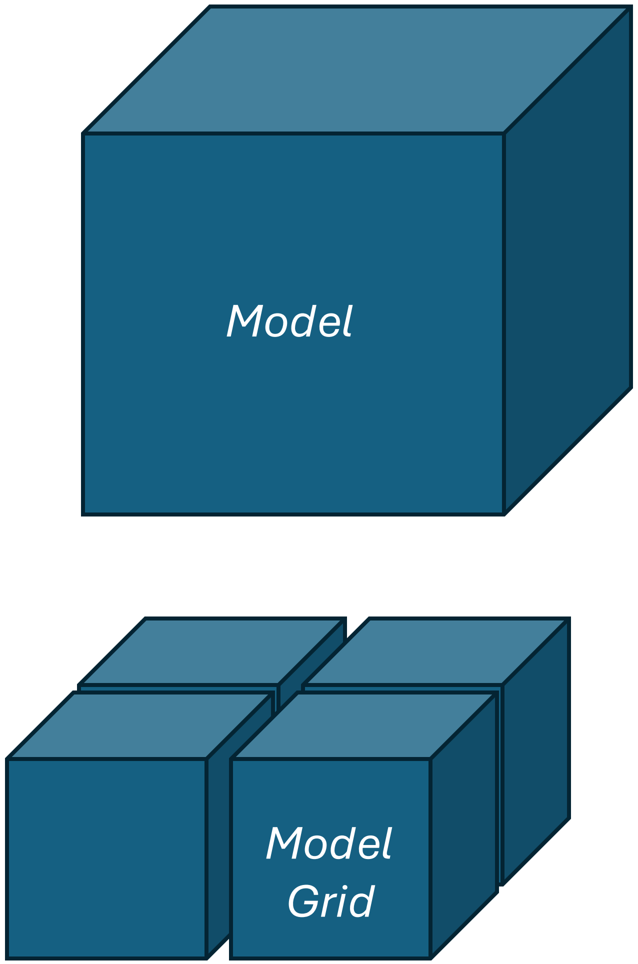
NEMO uses the commonly adopted Arakawa C-Grid (Arakawa and Lamb, 1977)

It also runs on Fortran!



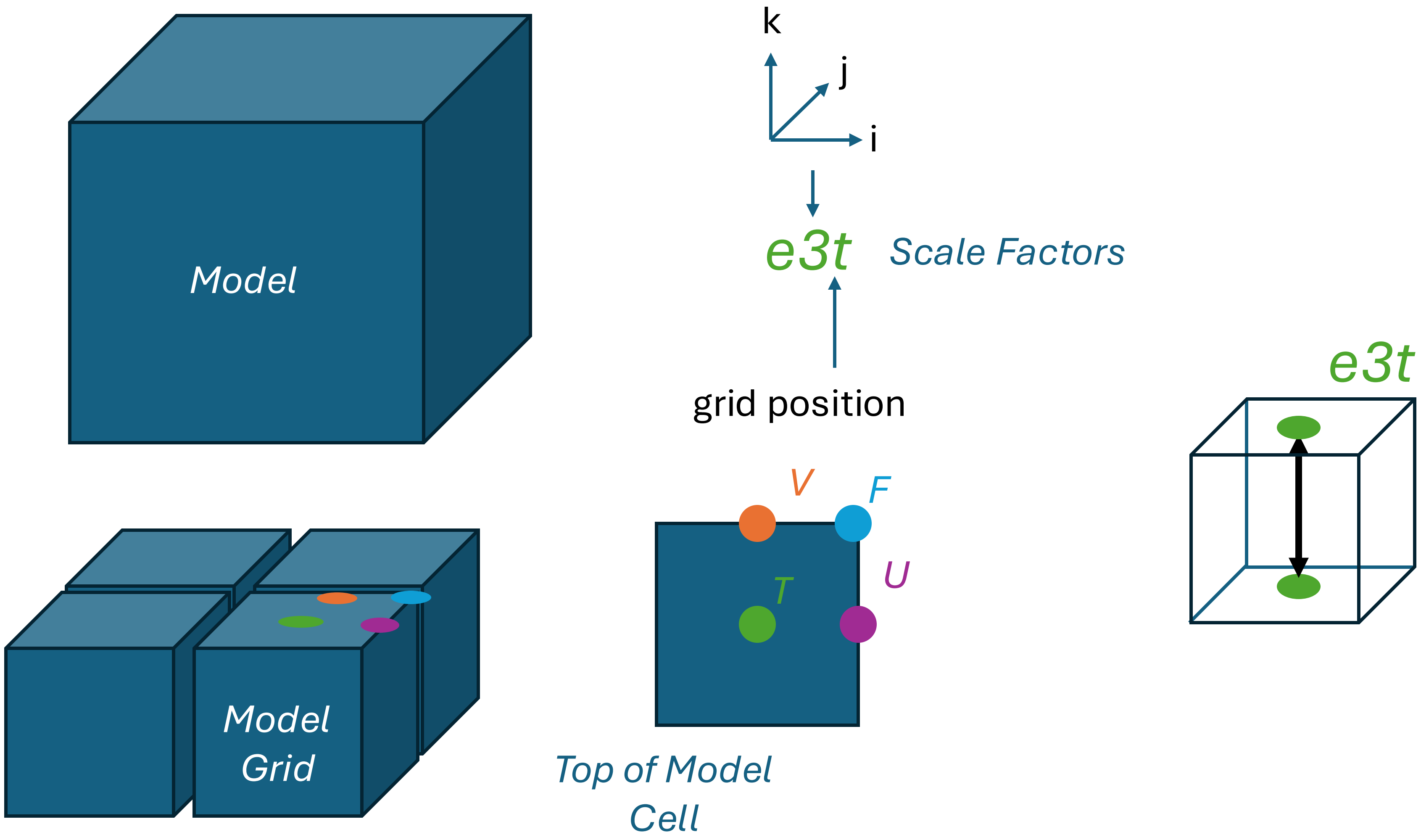
Challenge 1: Introduction

NEMO uses the commonly adopted Arakawa C-Grid (Arakawa and Lamb, 1977)



Challenge 1: Introduction

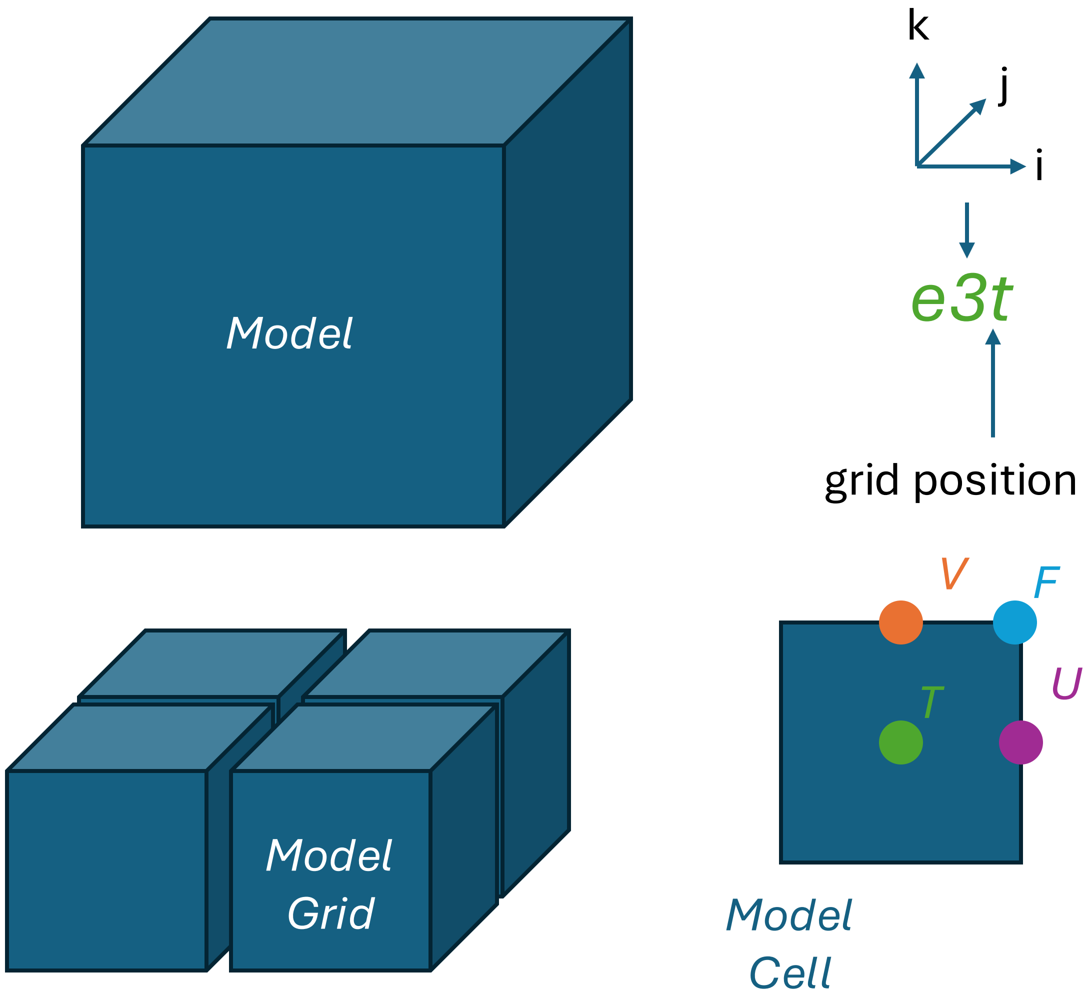
NEMO uses the commonly adopted Arakawa
C-Grid (Arakawa and Lamb, 1977)



VERIFY

Challenge 1: Introduction

NEMO uses the commonly adopted Arakawa C-Grid (Arakawa and Lamb, 1977)



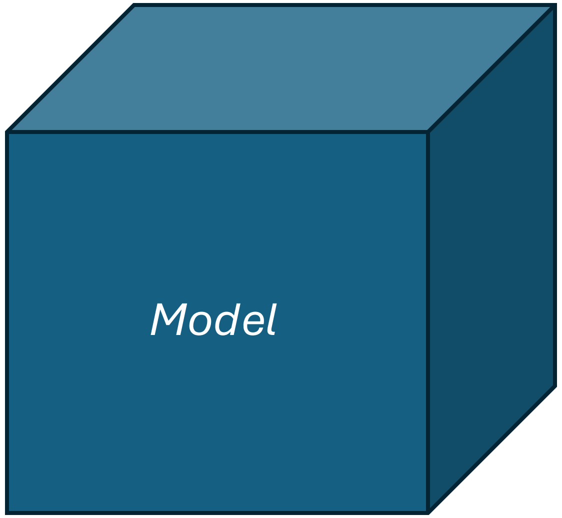
domain_cfg.nc

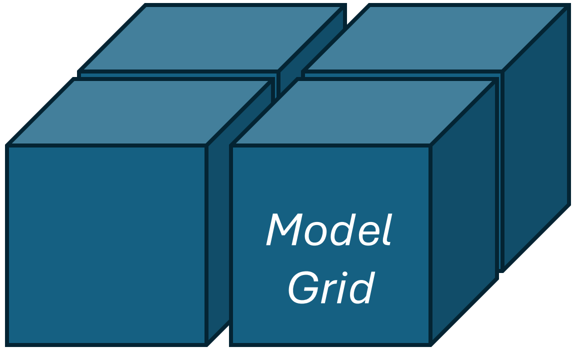
```
15]:  Data variables:
          bottom_level    (t, y, x) int32 640kB ...
          closea_mask     (y, x) float64 1MB ...
          e1f             (t, y, x) float64 1MB ...
          e1t             (t, y, x) float64 1MB ...
          e1u             (t, y, x) float64 1MB ...
          e1v             (t, y, x) float64 1MB ...
          e2f             (t, y, x) float64 1MB ...
          e2t             (t, y, x) float64 1MB ...
          e2u             (t, y, x) float64 1MB ...
          e2v             (t, y, x) float64 1MB ...
          e3f_0           (t, z, y, x) float64 96MB ...
          e3t_0           (t, z, y, x) float64 96MB ...
          e3t_1d          (t, z) float64 600B ...
          e3u_0           (t, z, y, x) float64 96MB ...
          e3uw_0          (t, z, y, x) float64 96MB ...
          e3v_0           (t, z, y, x) float64 96MB ...
          e3vw_0          (t, z, y, x) float64 96MB ...
          e3w_0           (t, z, y, x) float64 96MB ...
          e3w_1d          (t, z) float64 600B ...
          ff_f            (t, y, x) float64 1MB ...
          ff_t            (t, y, x) float64 1MB ...
          glamf           (t, y, x) float64 1MB ...
          glamt           (t, y, x) float64 1MB ...
          glamu           (t, y, x) float64 1MB ...
          glamv           (t, y, x) float64 1MB ...
          gphif           (t, y, x) float64 1MB ...
          gphit           (t, y, x) float64 1MB ...
          gphiu           (t, y, x) float64 1MB ...
          gphiv           (t, y, x) float64 1MB ...
          jperio          int32 4B ...
          jpiglo          int32 4B ...
```

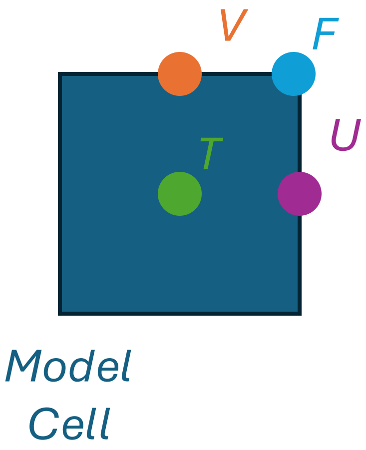
Scale Factors

Latitude and Longitude

Challenge 1: Introduction







T-Grid Data

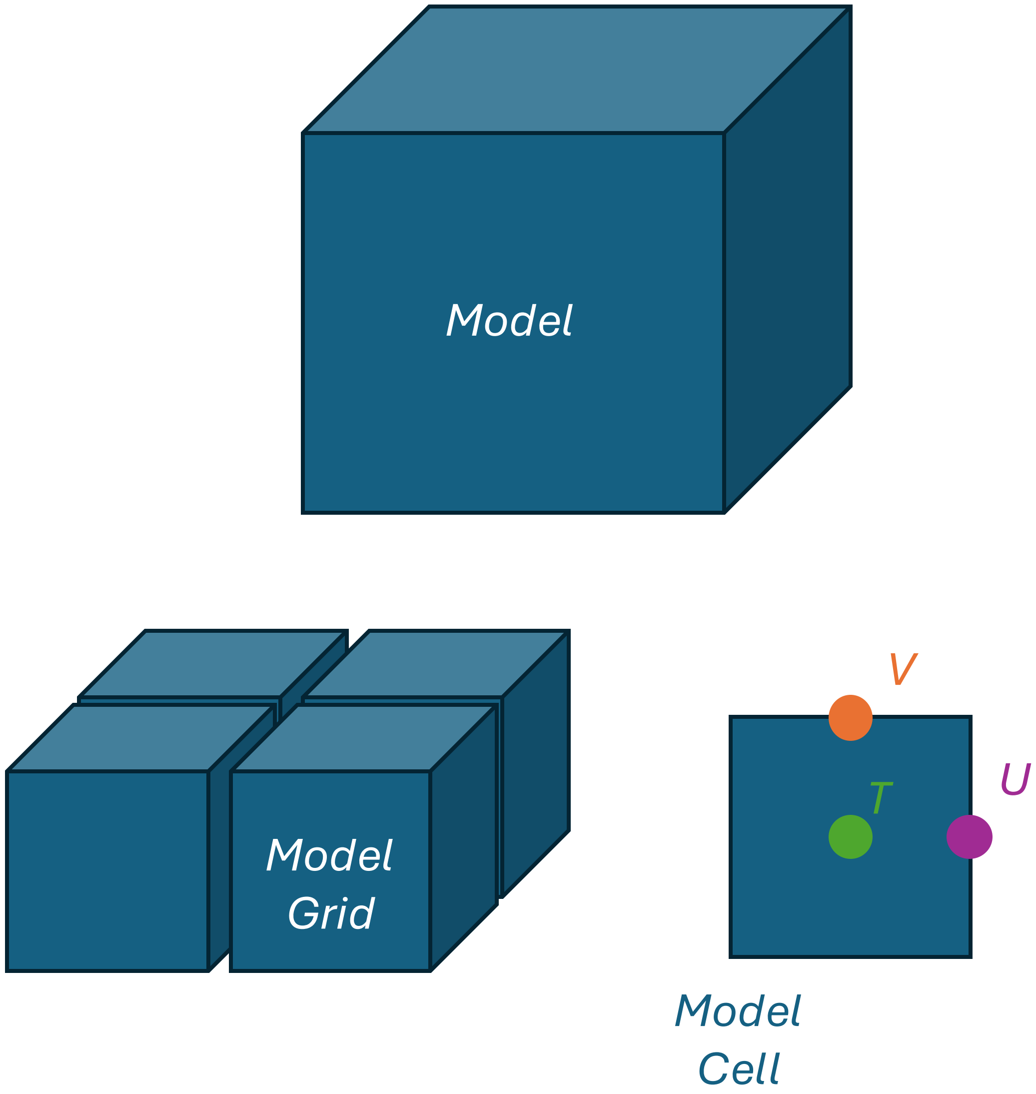
In [1]:
```python
import xarray as xr # xarray for accessing and manipulating data
```

In [2]:
```python
path = "/gws/pw/j07/workshop/ARIA_src_data/"
t_path = path + "VERIFY_eORCA025_MED_UKESM_19900101_20710101_grid_T.nc"
```

In [3]:
```python
ds = xr.open_dataset(t_path)
```

VERIFY

Challenge 1: Introduction



T-Grid Data

```
Out[4]: <xarray.Dataset> Size: 7GB
        Dimensions:              (time_counter: 972, y: 400, x: 400, axis_nbounds: 2)
        Coordinates:
            nav_lat              (time_counter, y, x) float32 622MB ...
            nav_lon              (time_counter, y, x) float32 622MB ...
            time_centered        (time_counter) object 8kB ...
          * time_counter         (time_counter) object 8kB 1990-01-16 00:00:00 ... 2...
        Dimensions without coordinates: y, x, axis_nbounds
        Data variables:
            mldr10_1             (time_counter, y, x) float32 622MB ...
            qns_oce              (time_counter, y, x) float32 622MB ...
            qsr_oce              (time_counter, y, x) float32 622MB ...
            sbt                  (time_counter, y, x) float32 622MB ...
            sos                  (time_counter, y, x) float32 622MB ...
            taum                 (time_counter, y, x) float32 622MB ...
            time_centered_bounds (time_counter, axis_nbounds) object 16kB ...
            time_counter_bounds  (time_counter, axis_nbounds) object 16kB ...
            tos                  (time_counter, y, x) float32 622MB ...
            wfo                  (time_counter, y, x) float32 622MB ...
            zos                  (time_counter, y, x) float32 622MB ...
        Attributes: (12/14)
            name:          MEANS_OUT/eORCA025_MED_UKESM_1m_19900101_19921230_grid_T
            description:   ocean T grid variables
            title:         ocean T grid variables
            Conventions:   CF-1.6
            timeStamp:     2021-Nov-06 06:11:48 GMT
            uuid:          89941c43-5b52-4f24-8d21-fff8d0897b4c
            ...            ...
            jbegin:        0
            nj:            76
            file_name:     eORCA025_MED_UKESM_1m_19900101_19921230_grid_T_199001-19900...
            TimeStamp:     06/11/2021 18:12:14 +0000
            history:       Wed Nov 12 13:39:58 2025: ncks -d x,801,1200 -d y,801,1200 ...
            NCO:           netCDF Operators version 5.3.3 (Homepage = http://nco.sf.ne...
```

VERIFY

Challenge 1: Introduction

- Selecting variables shows the associated meta data
- The example here is for shortwave radiation

```
ds.qsr_oce # we can inspect individual variables
```

```
<xarray.DataArray 'qsr_oce' (time_counter: 972, y: 400, x: 400)> Size: 622MB
[155520000 values with dtype=float32]
Coordinates:
    nav_lat        (y, x) float32 640kB ...
    nav_lon        (y, x) float32 640kB ...
    time_centered  (time_counter) object 8kB ...
  * time_counter   (time_counter) object 8kB 1990-01-16 00:00:00 ... 2070-12-...
Dimensions without coordinates: y, x
Attributes:
    standard_name:       net_downward_shortwave_flux_at_sea_water_surface
    long_name:           solar heat flux at ocean surface
    units:               W/m2
    online_operation:    average
    interval_operation:  1350 s
    interval_write:      1 month
    cell_methods:        time: mean (interval: 1350 s)
```

Challenge 1: Introduction

Horizontal mean, reducing
to 1D time series

```
qsr_mean = ds.qsr_oce.mean(["x","y"])
```

```
qsr_mean # the dimensions have been to collaped to time only
```

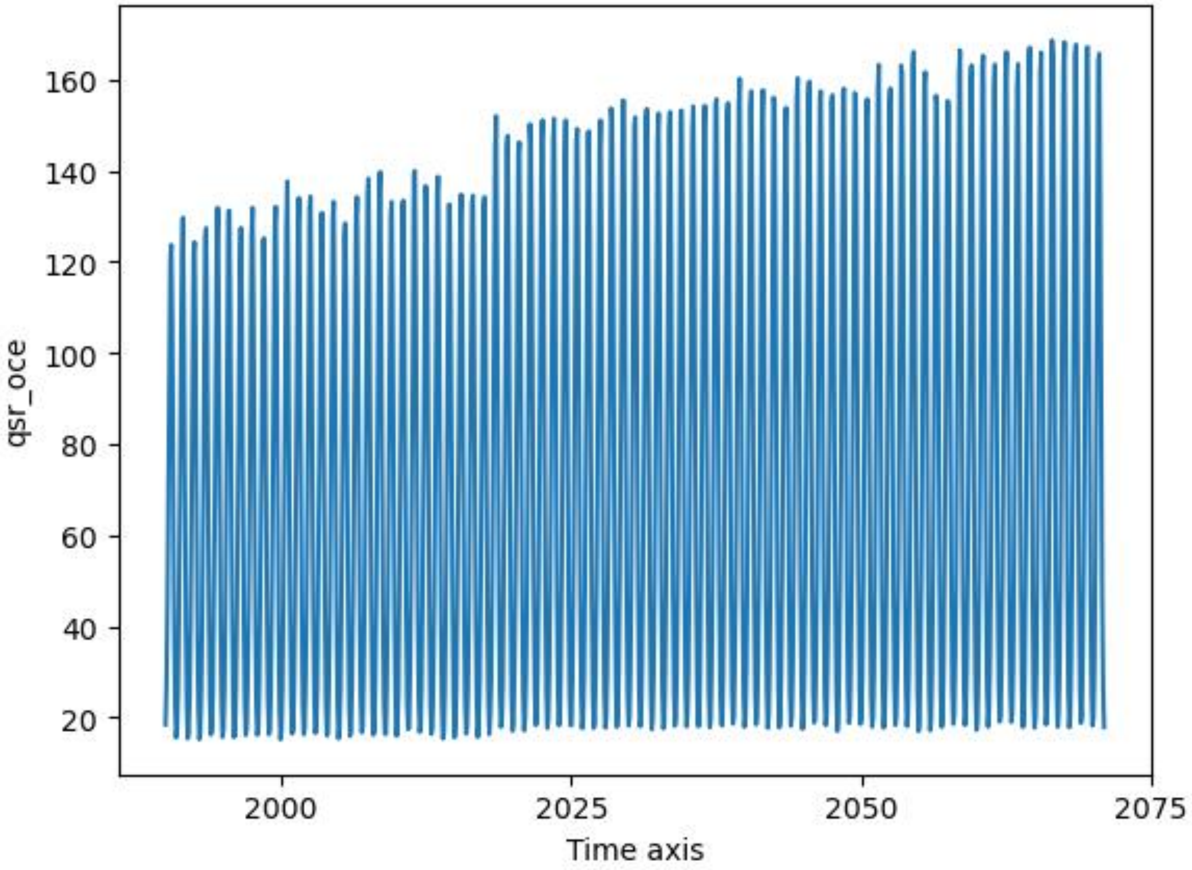
```
<xarray.DataArray 'qsr_oce' (time_counter: 972)> Size: 4kB
array([ 18.476593 ,  27.610086 ,  44.851322 ,  71.09863  ,  97.54053  ,
       120.826416 , 123.72689  , 106.80954  ,  71.47048  ,  40.029934 ,
        22.477606 ,  15.635142 ,  17.499926 ,  25.980236 ,  44.822742 ,
        68.59293  ,  98.48609  , 120.40601  , 129.7014   , 107.867874 ,
        70.8415   ,  40.07153  ,  22.752756 ,  15.448783 ,  16.896503 ,
        24.80583  ,  42.873085 ,  70.22497  ,  96.894714 , 121.878204 ,
       124.35743  , 105.791374 ,  69.4333   ,  39.713993 ,  21.998865 ,
        15.246516 ,  16.881447 ,  26.137096 ,  41.921658 ,  67.182686 ,
        99.329956 , 125.30074  , 127.40543  , 107.29668  ,  70.63323  ,
        40.50422  ,  23.103567 ,  16.292387 ,  17.892117 ,  25.44668  ,
        44.97561  ,  68.975296 ,  97.99222  , 125.18179  , 131.85834  ,
       111.15676  ,  71.27184  ,  40.056988 ,  22.876837 ,  15.727722 ,
        17.68961  ,  27.04288  ,  43.43357  ,  71.9569   , 100.051926 ,
       130.91718  , 131.25401  , 110.51976  ,  72.60136  ,  40.982906 ,
        22.747236 ,  15.689405 ,  17.004135 ,  27.034433 ,  44.861935 ,
        68.21526  ,  95.91902  , 119.6671   , 127.46469  , 108.8054   ,
        70.29683  ,  40.953907 ,  23.567652 ,  16.141476 ,  17.427288 ,
        27.914274 ,  44.219173 ,  67.68102  ,  96.98384  , 126.17667  ,
       131.86609  , 109.63617  ,  72.350426 ,  40.107918 ,  22.094315 ,
        16.225048 ,  17.533821 ,  27.340672 ,  45.542423 ,  69.96093  ,
...
```

Challenge 1: Introduction

Plotting of timeseries...

```
qsr_mean.plot() # the xarray plot function allows us to quickly plot data
```

```
[<matplotlib.lines.Line2D at 0x7efbdc5d1df0>]
```

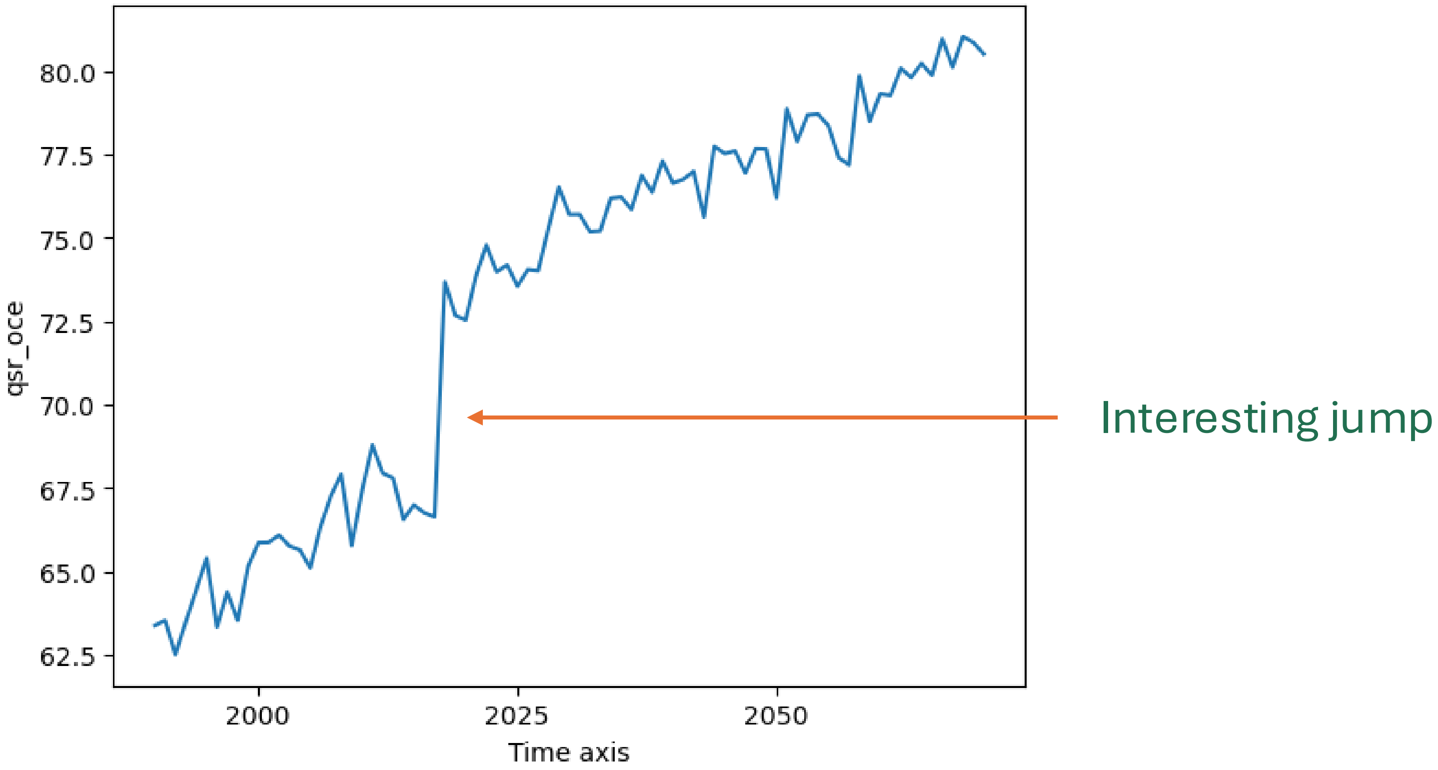


Challenge 1: Introduction

Using resample to extract a yearly mean of the timeseries

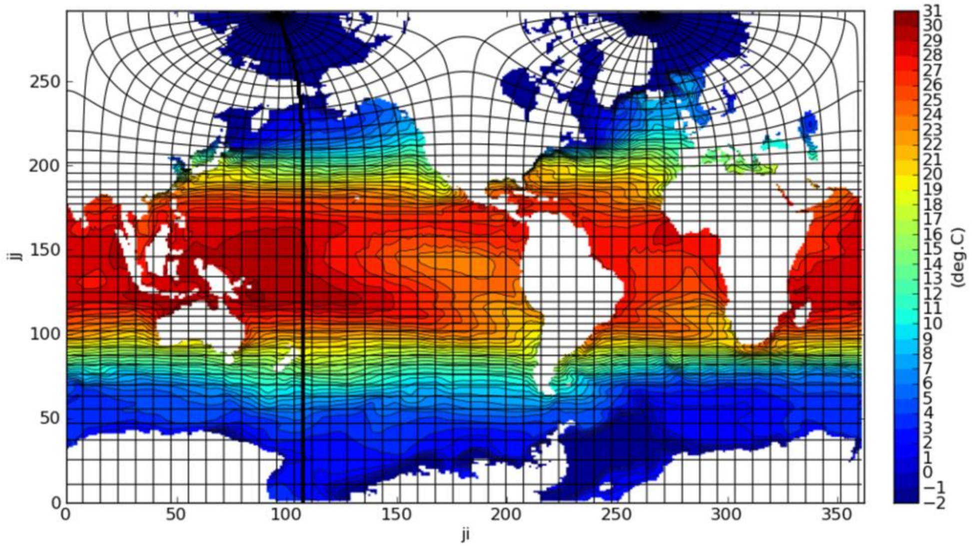
```
monthly_qsr = qsr_mean.resample(time_counter="YS").mean()
monthly_qsr.plot() # interesting
```

```
[<matplotlib.lines.Line2D at 0x7efbdb981df0>]
```



Challenge 1: Introduction

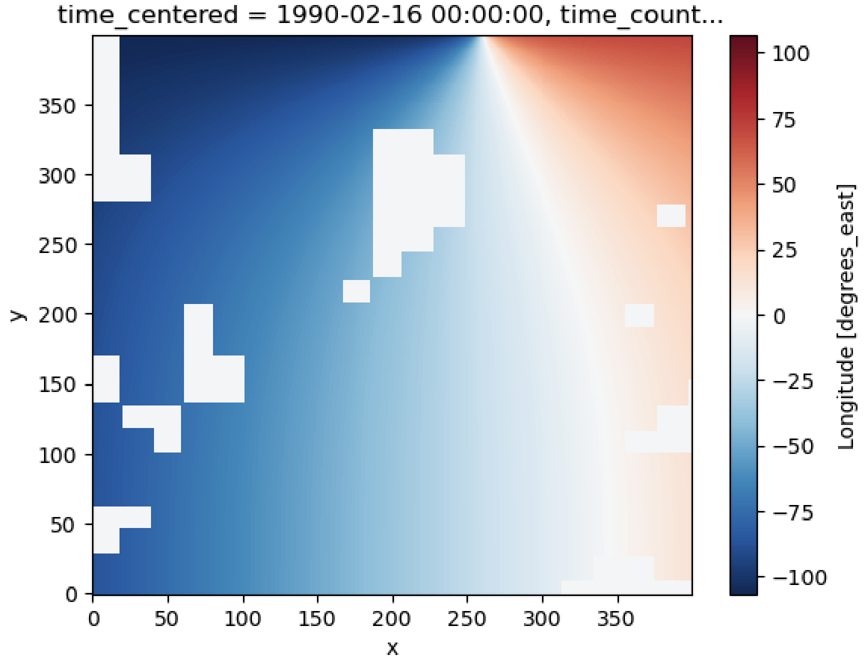
NEMO simulations commonly use a tri-polar grid, which distorts latitude and longitude.



https://brodeau.github.io/sosie/

```
ds.nav_lon.plot()
```

[10]: <matplotlib.collections.QuadMesh at 0x7f02e95a1dc0>

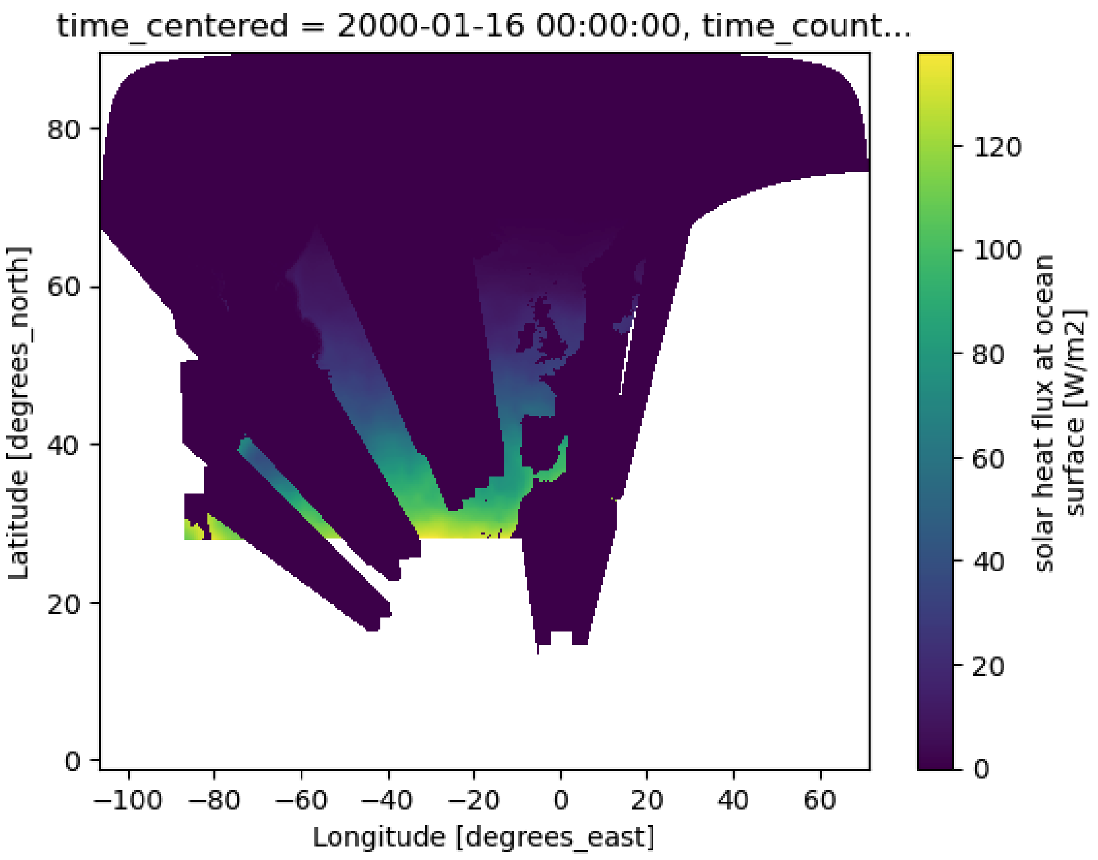


Challenge 1: Introduction

Extracting spatial 2D data

```
[25]: # first we select a date
      surf_rad_y2000m01 = ds.qsr_oce.sel(time_counter="2000-01")
```

```
[27]: # now we can plot...
      surf_rad_y2000m01.plot(x="nav_lon",y="nav_lat")
```

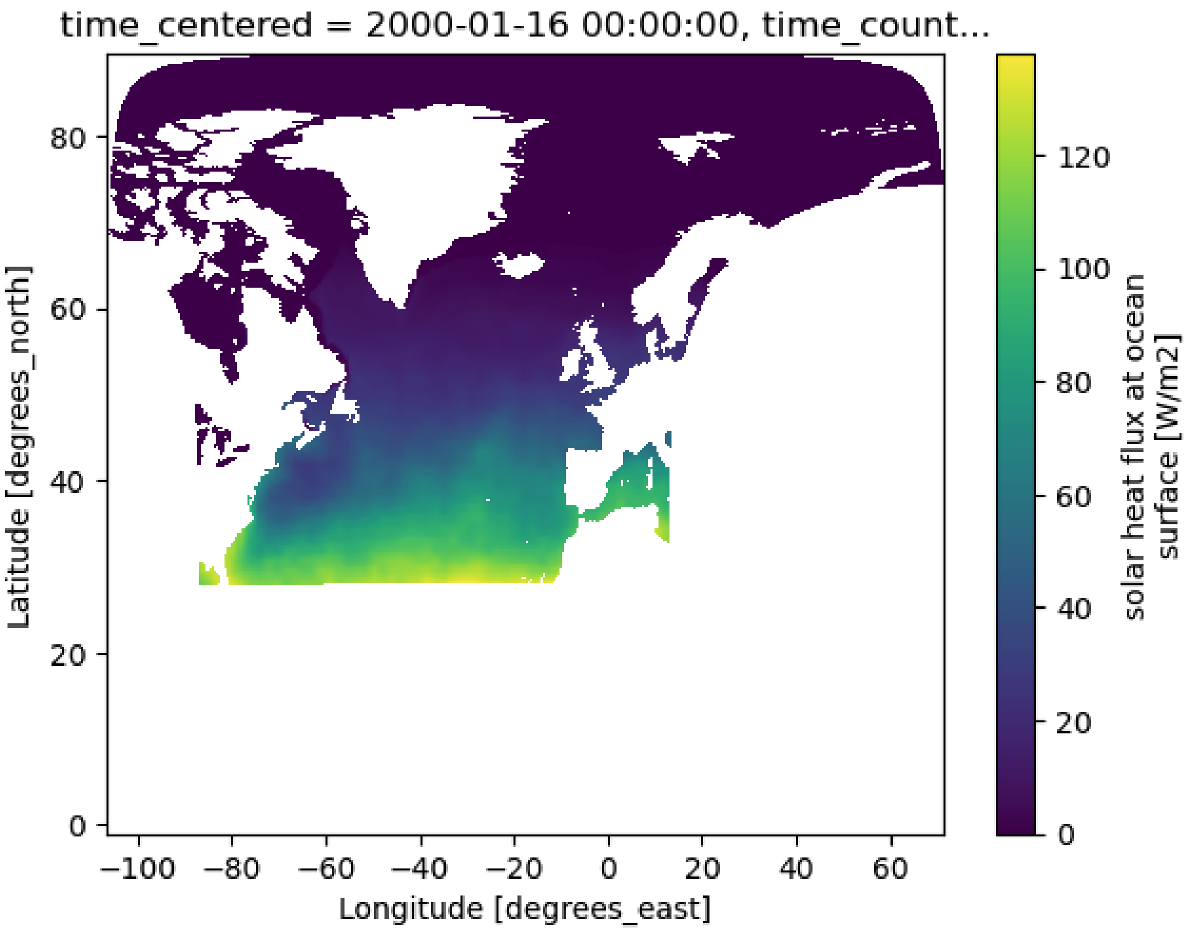


This looks rather peculiar. Masking of land is needed to avoid this.

VERIFY

Challenge 1: Introduction

```
[16]: surf_rad_y2000m01 = surf_rad_y2000m01.where(dom_cfg.top_level == 1)
      surf_rad_y2000m01.plot(x="nav_lon",y="nav_lat")
```

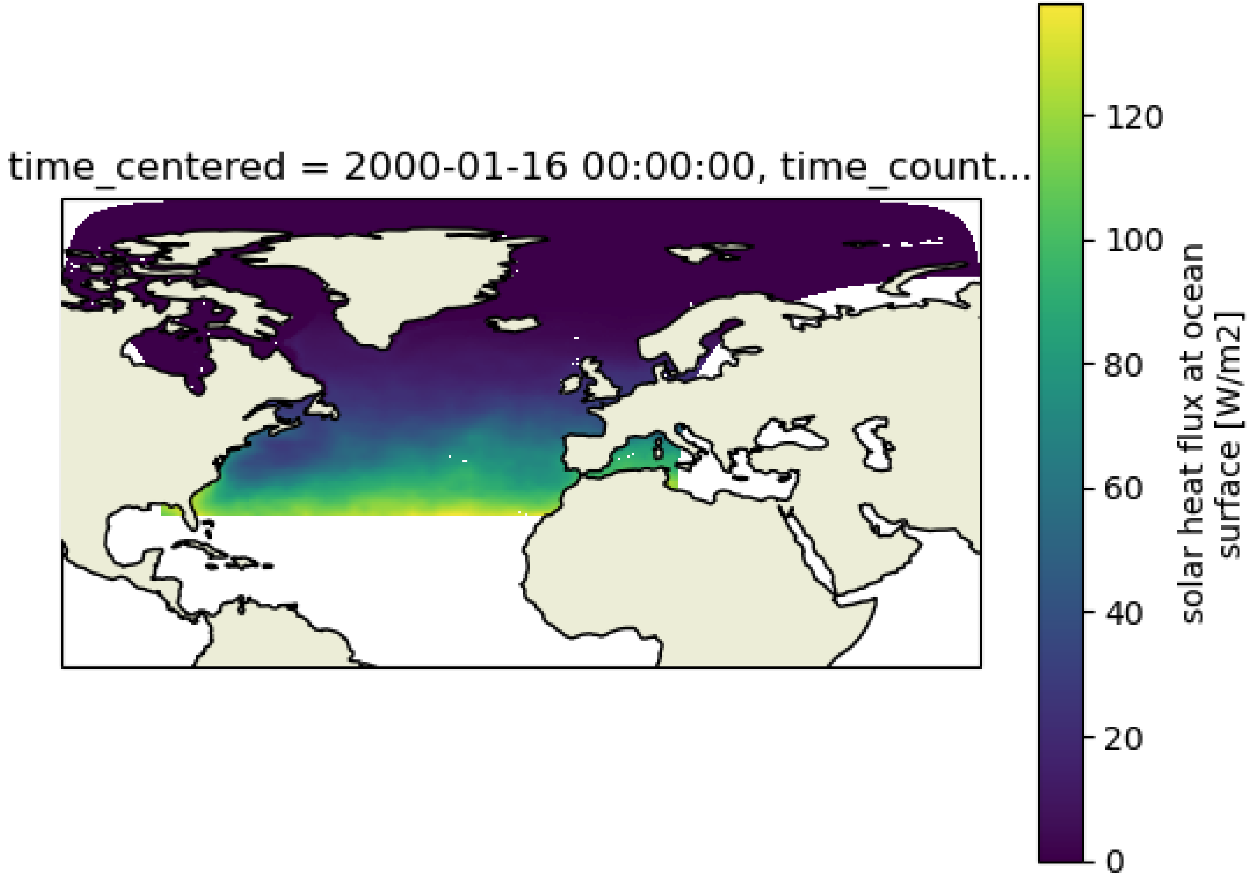


Result after masking of land

Challenge 1: Introduction

We can do fancier projections with cartopy

[17]:
```
# now let us plot using a transform
ax=plt.subplot(projection=ccrs.PlateCarree())
ax.add_feature(cfeature.LAND, zorder=100, edgecolor='k')
surf_rad_y2000m01.plot(ax=ax, x="nav_lon",y="nav_lat", transform=ccrs.PlateCarree())
```



VERIFY

Challenge 1: Introduction

Groups!

Group 1	Group 2	Group 3	Group 4
Anna Hardisty	Ella Beaven	Jack Wharton	Shivendra Singh Verma
Nina Brendling	Guy Ludford	Ellie Fisher	Evan Hambly
Thanasis Giannakopoulos	Molly Hammond	Ella Tanner	Brad Neimann
Isaac Foreman	Alejandro Coca-Castro	Andrea Quintanilla	Gaby Johnson
Amethyst Eicher	Aoife Ní Bhuachalla	Charli Frisby	Valeria Mascolo

VERIFY

Challenge 1: Introduction

Now over to you…

You have access to:

verify.domcfg.nc

VERIFY_eORCA025_MED_UKESM_19900101_20710101_grid_T.nc

VERIFY_eORCA025_MED_UKESM_19900101_20710101_icemod.nc

VERIFY_eORCA025_MED_UKESM_3D_19900101_20710101_grid_T.nc

VERIFY_eORCA025_MED_UKESM_3D_19900101_20710101_grid_U.nc

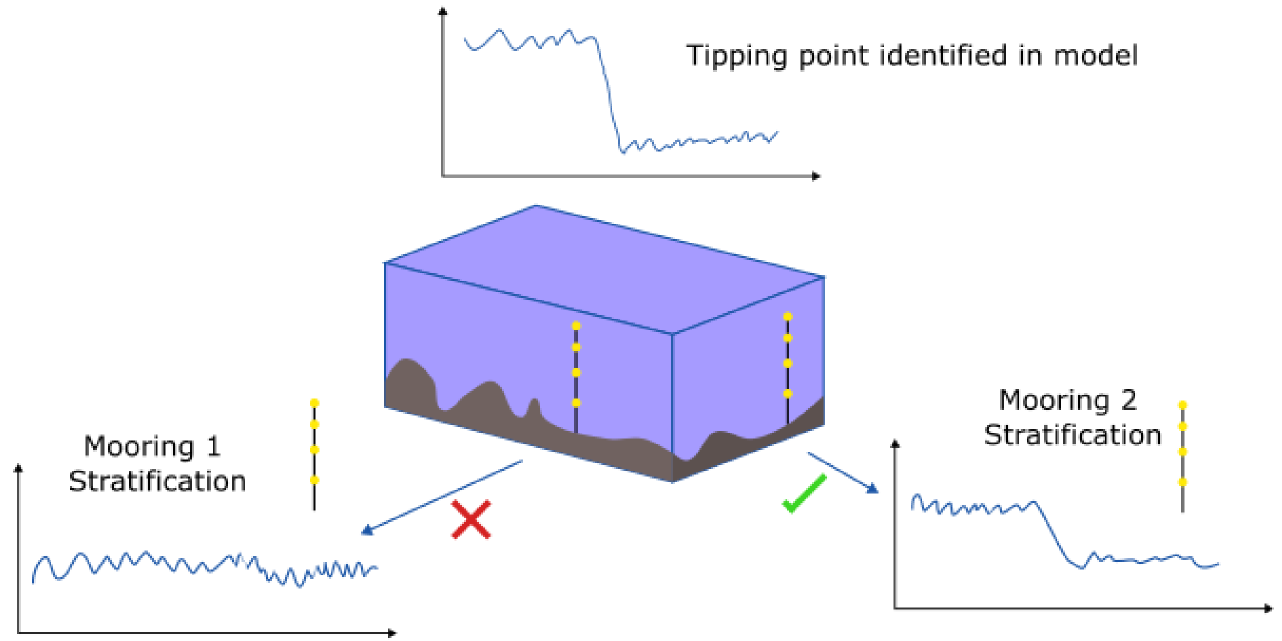
VERIFY_eORCA025_MED_UKESM_3D_19900101_20710101_grid_V.nc

Challenge 2

Challenge 2: Virtual Observations

"Demonstrate how one might devise an observational campaign to capture this tipping point"

Mooring Example



Challenge 2: Virtual Observations

3D dataset

Note the differences in variable format

```
# Your dataset path and  filename are /gws/pw/j07/workshop/ARIA_src_data/
# VERIFY_eORCA025_MED_UKESM_3D_19900101_20710101_grid_T.nc

import xarray as xr

path = "/gws/pw/j07/workshop/ARIA_src_data/"
t_path3d = path + "VERIFY_eORCA025_MED_UKESM_3D_19900101_20710101_grid_T.nc"

ds = xr.open_dataset(t_path3d, chunks={"x":400,"y":400})

ds
```

xarray.Dataset

► Dimensions: (**time_counter**: 852, **deptht**: 75, axis_nbounds: 2, y: 400, x: 400)

▼ Coordinates:

deptht	(deptht)	float32	0.5058 1.556 ... 5.902e+03
nav_lat	(y, x)	float32	dask.array<chunksize=(400, 400...
nav_lon	(y, x)	float32	dask.array<chunksize=(400, 400...
time_centered	(time_counter)	object	dask.array<chunksize=(852,), m...
time_counter	(time_counter)	object	1990-01-16 00:00:00 ... 2070-12-...

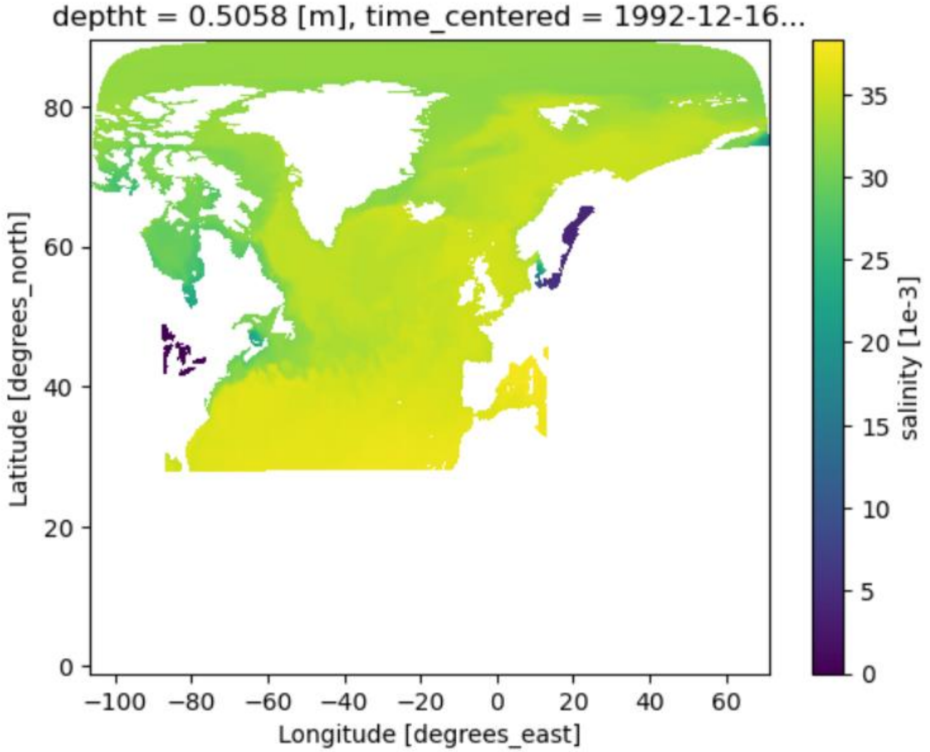
VERIFY

Challenge 2: Virtual Observations

You can change the depth you are looking at.

Take a 2d slice at one time
Eg. surface

Look at it over the 2d field.
This is similar to challenge1.




```
# Select a date and a depth level
# Use the sel() function to select the data based on values of a specific dimension.
# Use the isel() function to select the data based on the index position.

# This is an example for december 1992:
# 3D salinity field for that month: (deptht, y, x)
so_3d = ds.so.sel(time_counter="1992-12")

# 2D field for the same time at a specific depth level (deptht dimension). Change the deptht
Salinity = ds.so.sel(time_counter="1992-12").isel(deptht=0)
# alternatives:
# Salinity = ds.so.sel(time_counter="1992-12").sel(deptht=0, method="nearest") # in this case
# Salinity = ds.so.isel(time_counter=35).isel(deptht=0) # in this case the time is not the d

# mask the data to cover the land regions using the domain file.
dom_cfg = xr.open_dataset(path + "verify.domcfg.nc") # open the domain
Salinity = Salinity.where(dom_cfg.top_level ==1) # apply maks

# Finally plot your salinity.
Salinity.plot(x="nav_lon", y="nav_lat")
```

VERIFY

Challenge 2: Virtual Observations

Generate a mask using the salinity field. (this is a hack – usually the mask would be generated with the model outputs)

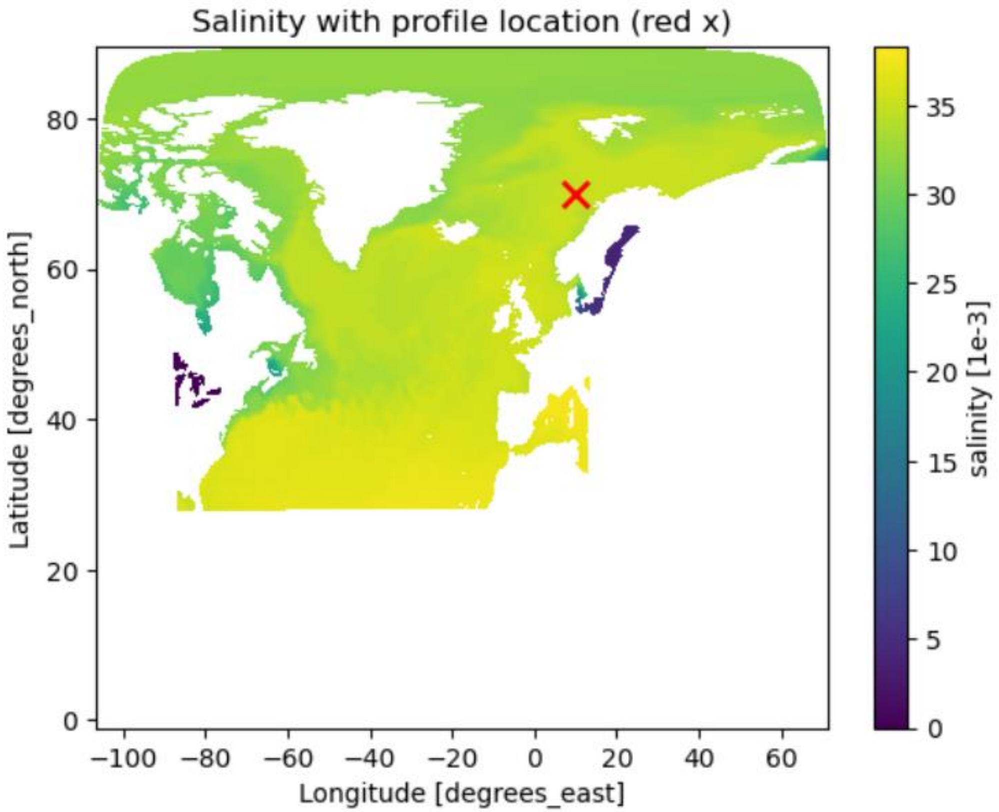
```
# Mask for the full dataset. 4D (time_counter, deptht, y, x)
mask = xr.where(ds.so == 0, float("nan"), 1.0)
```

Select one point in space (lat/lon) and time (time_counter) to look at all depth.
First find the closest model point to your coordinates, the get all depth at one time.

```
# One more recomendable way to pick a point is to first select a latitutde and longitude you want to investigate,
target_lat = 70
target_lon = 10

# Then find the closest model point to these coordinates
dist = ((ds.nav_lat - target_lat)**2 + (ds.nav_lon - target_lon)**2)**0.5

# Find the index of the closest point by:
dist_1d = dist.stack(points=("y", "x")).load() # Turning the 2D grid (y, x) into a 1D list of points called "point
ip = dist_1d.argmin("points") # getting index of the point with minimum distance (closest point)

# Recover the original (y, x) indices from the stacked MultiIndex
j = dist_1d["y"].isel(points=ip).item()
i = dist_1d["x"].isel(points=ip).item()

#Use (j, i) with isel to get the vertical profile (all depths, one time)
salinity_profile = ds.so.sel(time_counter="1992-12").isel(y=j, x=i)

#print(salinity_profile)
#print(salinity_profile.deptht.values)
```

Challenge 2: Virtual Observations

Check the point is where you wanted it

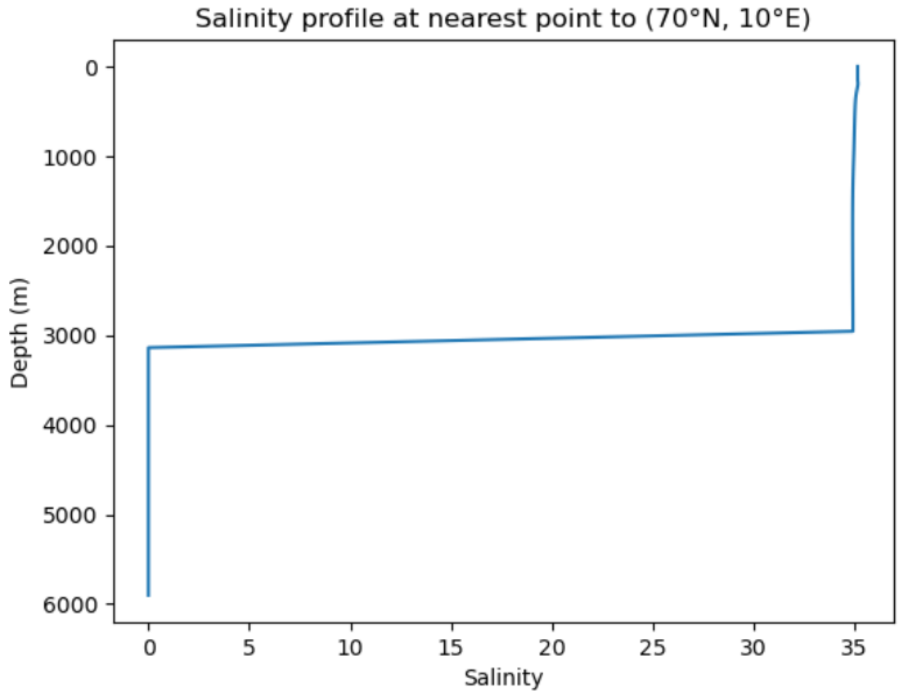



```python
import matplotlib.pyplot as plt

# Lon/lat of the chosen point
pt_lon = ds.nav_lon.isel(y=j, x=i)
pt_lat = ds.nav_lat.isel(y=j, x=i)

# Plot Salinity and overlay the red 'x'
fig, ax = plt.subplots()

Salinity.plot(x="nav_lon", y="nav_lat", ax=ax)
ax.plot(pt_lon, pt_lat, "rx", markersize=10, mew=2)

ax.set_title("Salinity with profile location (red x)")
```

Challenge 2: Virtual Observations

Plot your salinity profile un-masked

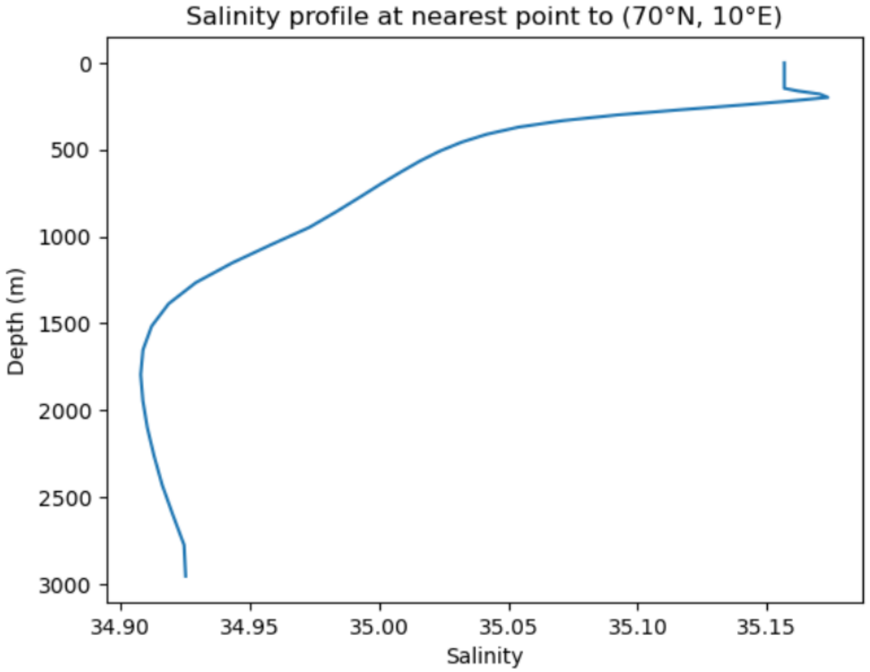
```
# Simple profile: salinity vs depth
# import matplotlib.pyplot as plt

salinity_profile.plot(y="deptht")   # or x="deptht" if you prefer depth on x-axis
plt.gca().invert_yaxis()            # optional: depth increasing downward
plt.title("Salinity profile at nearest point to (70°N, 10°E)")
plt.xlabel("Salinity")
plt.ylabel("Depth (m)")
```



Challenge 2: Virtual Observations

Plot your salinity profile masked

```
# Extract the 1D mask profile at the same time and point
mask_profile = mask.sel(time_counter="1992-12").isel(y=j, x=i)

# Apply the mask: keep values where mask == 1, set others to NaN
salinity_profile_masked = salinity_profile.where(mask_profile == 1)

# Plot masked profile
salinity_profile_masked.plot(y="deptht")   # or x="deptht"
plt.gca().invert_yaxis()
plt.title("Salinity profile at nearest point to (70°N, 10°E)")
plt.xlabel("Salinity")
plt.ylabel("Depth (m)")
```

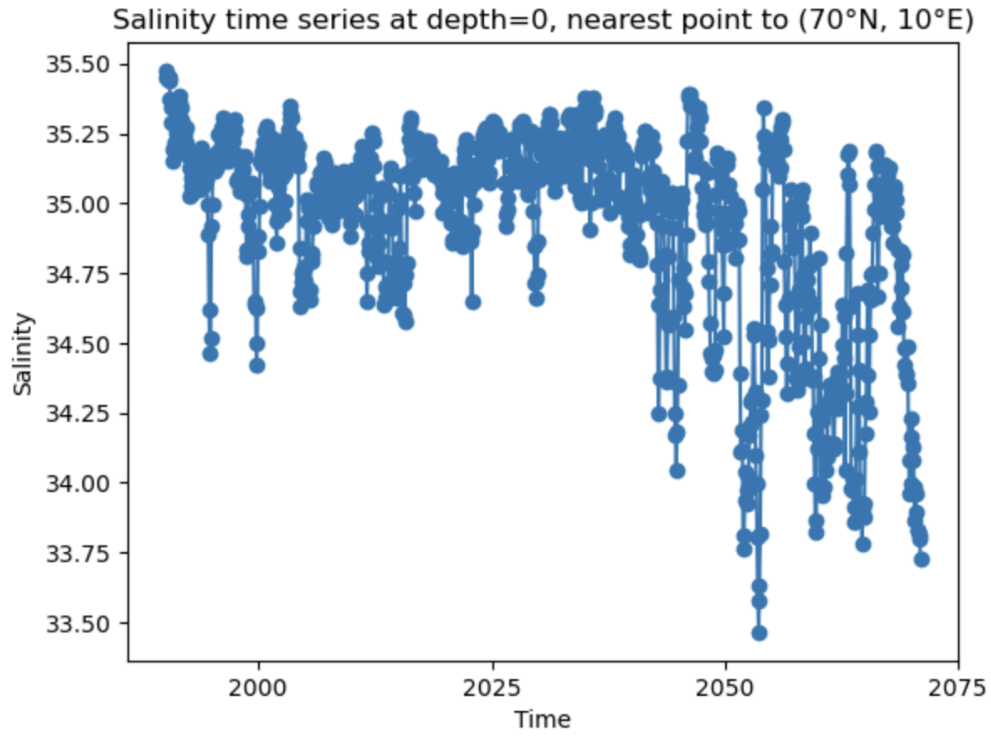


VERIFY

Challenge 2: Virtual Observations

Try a timeseries:
Same point,
one depth,
all times.

```python
# Time series at surface (deptht=0) for the selected grid point (j, i)
salinity_ts = ds.so.isel(deptht=0, y=j, x=i)   # dims: time_counter

# plot the timeseries
#import matplotlib.pyplot as plt
fig, ax = plt.subplots()

# Line + scatter time series
salinity_ts.plot(ax=ax)
ax.scatter(salinity_ts["time_counter"].values,
           salinity_ts.values)

ax.set_title("Salinity time series at depth=0, nearest point to (70°N, 10°E)")
ax.set_xlabel("Time")
ax.set_ylabel("Salinity")

plt.show()
```

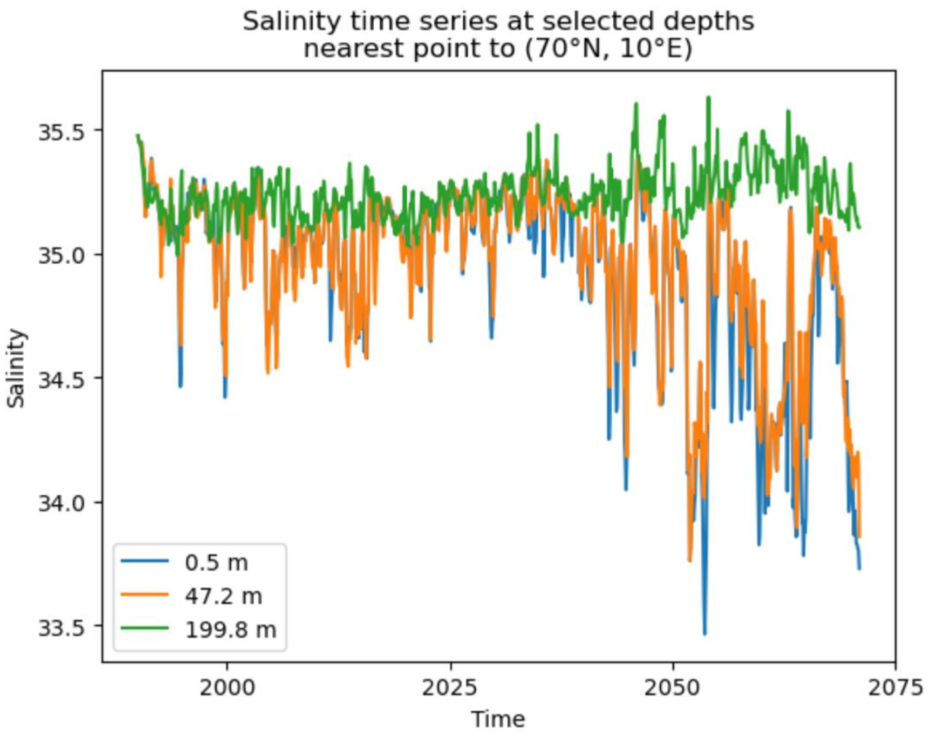


VERIFY

Challenge 2: Virtual Observations

You can do timeseries at different depths

```
# Hand-picked depths in metres
depths = [0, 50, 200]    # change as you like

fig, ax = plt.subplots()
for d in depths:
    # time series at depth ~ d, for point (j, i)
    salinity_ts = ds.so.sel(deptht=d, method="nearest").isel(y=j, x=i) # find the closest
    label = f"{float(salinity_ts.deptht.values):.1f} m"     # label with the actual depth
    salinity_ts.plot(ax=ax, label=label)

ax.set_title("Salinity time series at selected depths\nnearest point to (70°N, 10°E)")
ax.set_xlabel("Time")
ax.set_ylabel("Salinity")
ax.legend()
plt.show()
```



VERIFY

Challenge 2: Virtual Observations

But a better way to look at one profile over multiple times is a *Hovmöller plot* .

First select all depth at one point for the full timeseries subset and mask it.

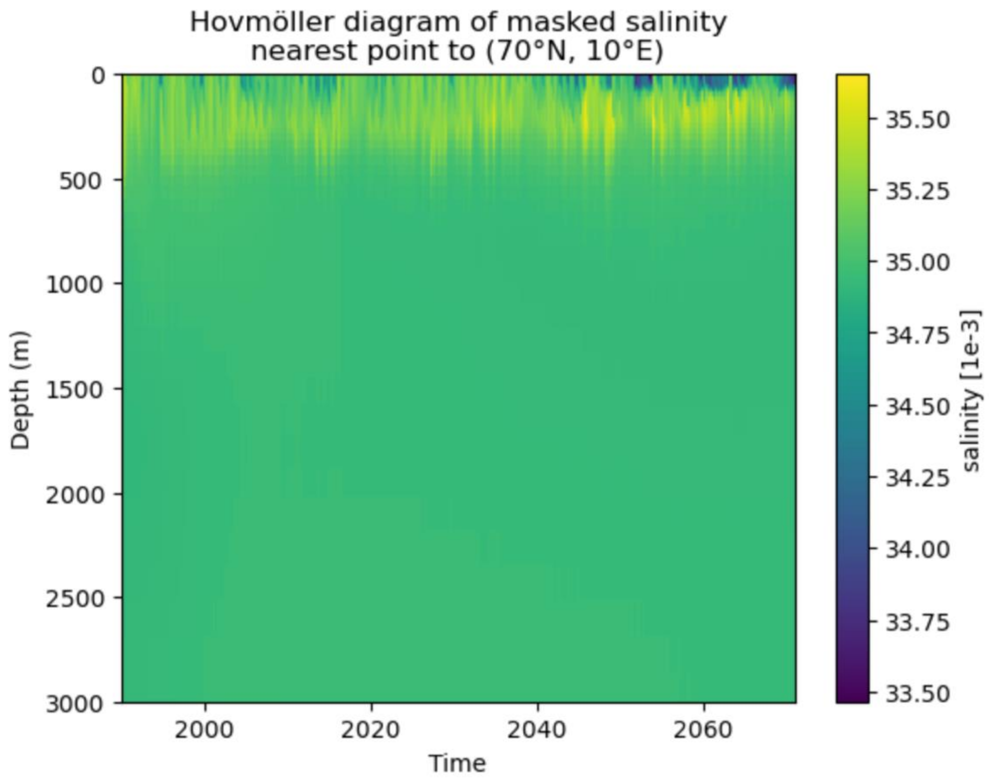
```
# Extract salinity and mask at the chosen horizontal point (j, i)
salinity_hov = ds.so.isel(y=j, x=i)   # dims: ('time_counter', 'deptht')
mask_hov     = mask.isel(y=j, x=i)    # same dims. This uses the 4D mask you have generated at

# Apply mask to you salinity slice.
salinity_hov_masked = salinity_hov.where(mask_hov == 1)

# (optional sanity check)
print(salinity_hov_masked.dims)       # should be ('time_counter', 'deptht')
```

VERIFY

Challenge 2: Virtual Observations

Then plot your *Hovmöller plot* .

```python
# Hovmöller plot: time vs depth
fig, ax = plt.subplots()

salinity_hov_masked.plot(
    x="time_counter",
    y="deptht",
    ax=ax,
)
# x-limits: first and last time (you can change these)
t0 = salinity_hov_masked["time_counter"].values[0]
t1 = salinity_hov_masked["time_counter"].values[-1]
ax.set_xlim(t0, t1)
# y-limits: e.g. top 500 m (and flip so 0 is at the top)
ax.set_ylim(3000, 0)

ax.set_title("Hovmöller diagram of masked salinity\nnearest point to (70°N, 10°E)")
ax.set_xlabel("Time")
ax.set_ylabel("Depth (m)")
```



VERIFY

Challenge 2: Virtual Observations

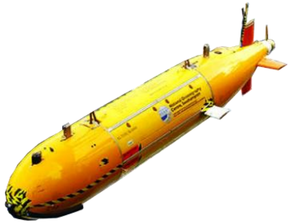
Now over to you...

Play around, try different points, and different subsets of data...

Can you see a tipping point?

What about testing different types of observations? Ship-based underway or saildrones sampling perhaps?

Autosub



Ship-based underway

Unmanned surface vehicles

VERIFY

Challenge 3:
Advanced

Challenge 3: Tipping Point detection

What we will cover

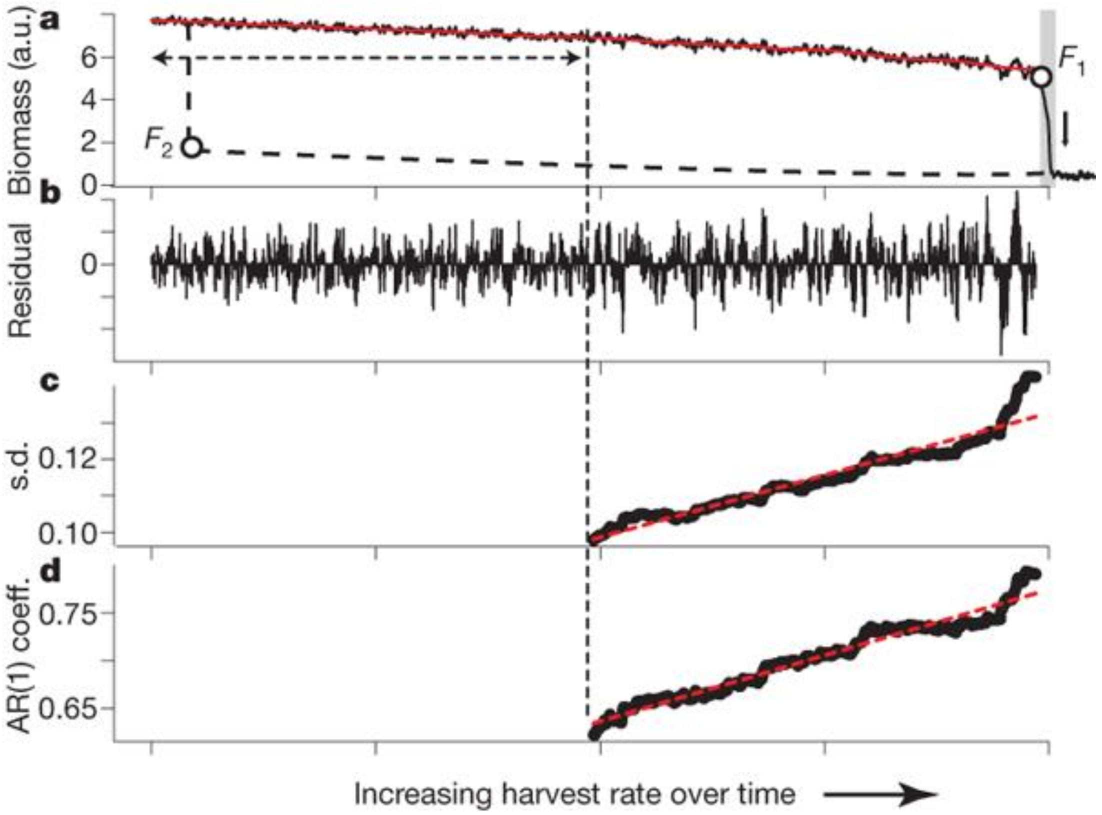
- We will introduce methods for tipping point detection
- Key metrics include variance and auto-correlation

Challenge 3: Tipping Point detection

Tipping Point Overview

- Systems can behave differently in the lead up to a tipping point
- Statistical analysis can reveal the changes

1. Raw timeseries
2. Detrended timeseries
3. Variance$^{1/2}$
4. Lag-1 Autocorrelation

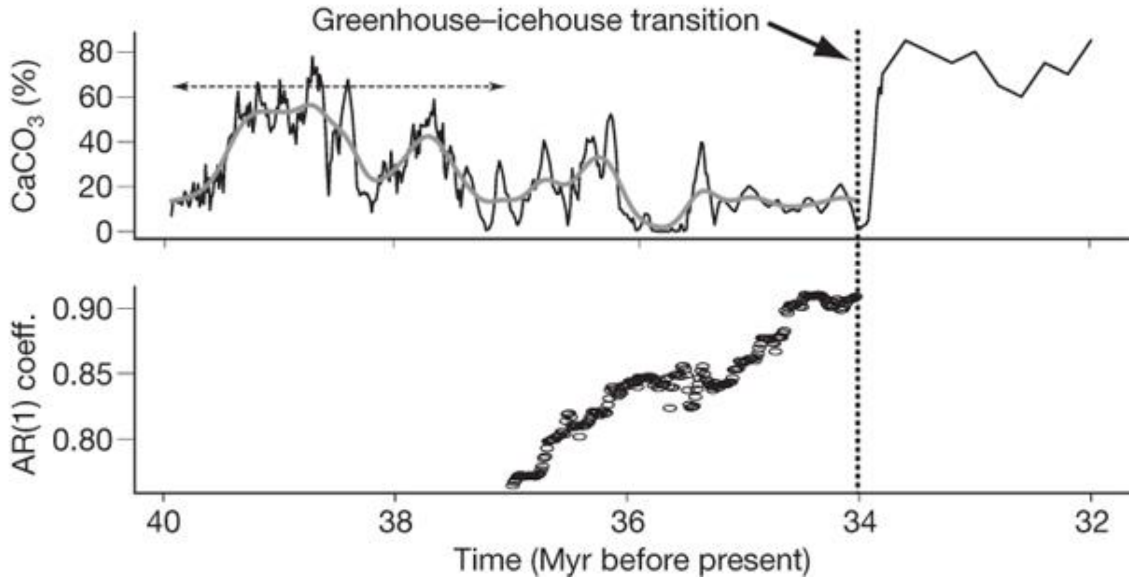


Scheffer et al. (2009)

Challenge 3: Tipping Point detection

Example in palaeo record

- We observe an abrupt transition in the percentage of $CaCO_3$
- Associated rise on auto-correlation



Scheffer et al. (2009)

Challenge 3: Tipping Point detection

Worked example

- Example with surface temperature extraction
- Extract a location as before

```
path = "/gws/pw/j07/workshop/ARIA_src_data/"
t_path = path + "VERIFY_eORCA025_MED_UKESM_19900101_20710101_grid_T.nc"
```

```
tos = xr.open_dataset(t_path).tos # get surface temperature data
```

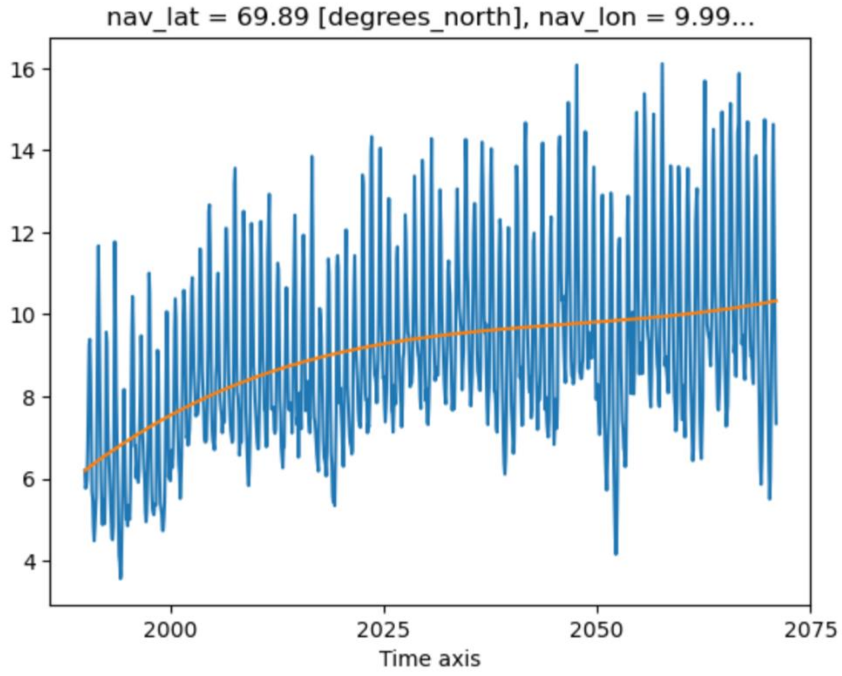
```
# Extract a time series but choosing a latitutde and longitude you want to investig
target_lat = 70
target_lon = 10

# Then find the closest model point to these coordinates
dist = ((tos.nav_lat - target_lat)**2 + (tos.nav_lon - target_lon)**2)**0.5

# Find the index of the closest point by:
dist_1d = dist.stack(points=("y", "x")) # Turning the 2D grid (y, x) into a 1D list
ip = dist_1d.argmin("points") # getting index of the point with minimum distance (c

# Recover the original (y, x) indices from the stacked MultiIndex
j = dist_1d["y"].isel(points=ip).item()
i = dist_1d["x"].isel(points=ip).item()

tos_timeseries = tos.isel(x=i,y=j)
```

VERIFY

Challenge 3: Tipping Point detection

We then detrend the data by fitting a curve to the timeseries and removing the fitted “trend”

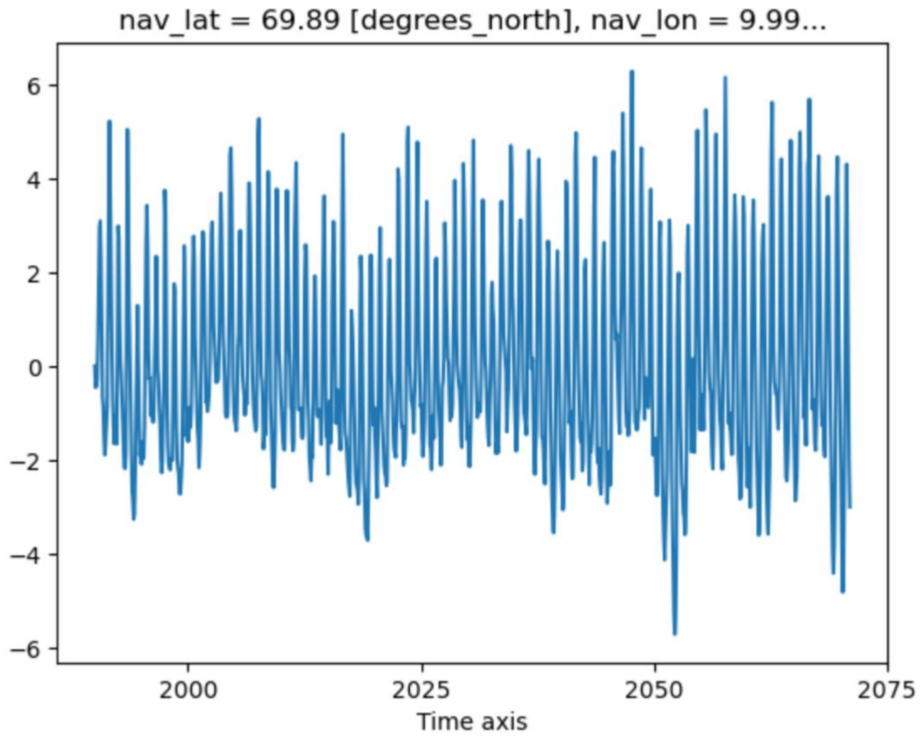
```
# This function is designed to remove the trend from the timeseries
def detrend_dim(da, dim, deg=1):
    # detrend along a single dimension
    p = da.polyfit(dim=dim, deg=deg)
    fit = xr.polyval(da[dim], p.polyfit_coefficients)
    return da - fit, fit

# detrended_tos is the detrended temperature timeseries and fit defined the fitted curve
detrended_tos, fit = detrend_dim(tos_timeseries, "time_counter", 3)

# we can show the fit here
tos_timeseries.plot()
fit.plot()
```



Challenge 3: Tipping Point detection

Detrended result

```python
# once detrended, we recover temporal variability
detrended_tos.plot()
```

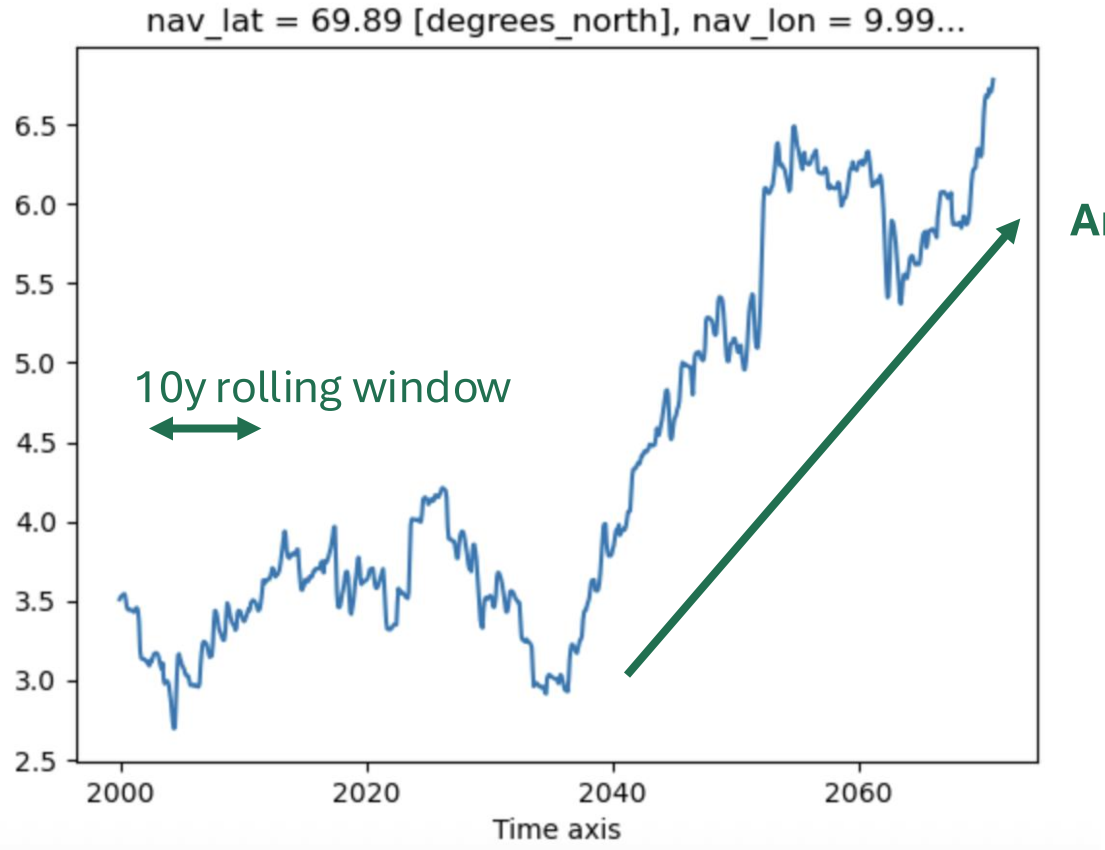


Challenge 3: Tipping Point detection

We then find the variance over a rolling window – ten years in this case

```
# let's calcualte variance over a rolling window
tos_variance = detrended_tos.rolling(time_counter=120).var()
```

```
# We then plot the variance to evaluate the emergence of a tipping point in this system
tos_variance.plot()
```



Are we heading for a tipping point!?

VERIFY

Challenge 3: Tipping Point detection

Over to you

- Use the notebooks as a guide (see additional autocorrelation example)
- Try out different timeseries data. What about Challenge 1 data?
- Can you think of other methods that might be useful for detecting tipping?