Climate models for ecologist

Isaac Brito-Morales

2021-09-02

Contents

Chapter 1

Welcome

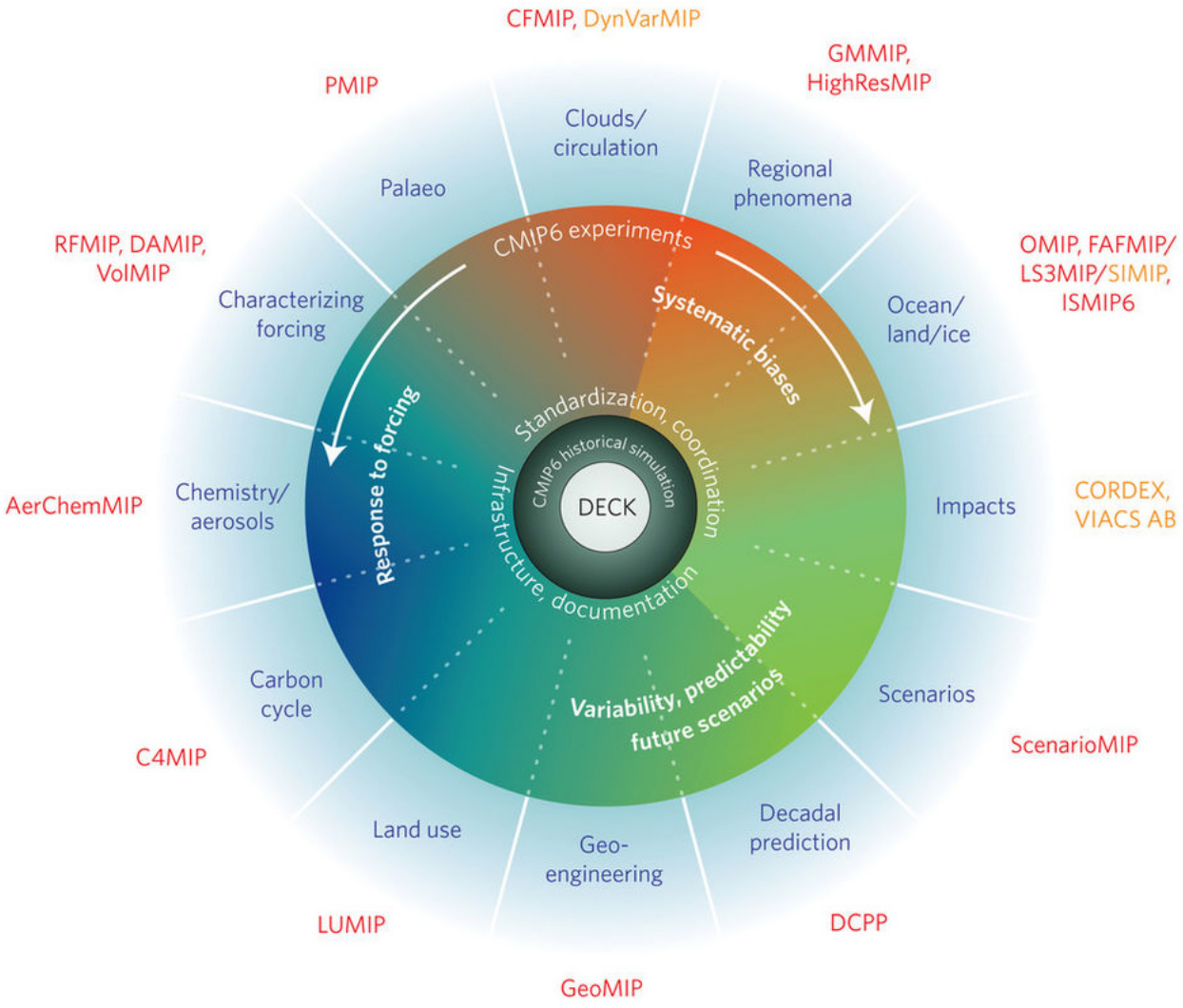
This document has for objective explain how to process and calculate climate change models to estimate climate change metrics.

Chapter 2

CMIP6 models

CMIP6 stands for Coupled Model Intercomparison Project (CMIP). These are climate models that simulate the physics, chemistry and biology of the atmosphere in detail (some of them!). The 2013 IPCC fifth assessment report (AR5) featured climate models from CMIP5, while the upcoming 2021 IPCC sixth assessment report (AR6) will feature new state-of-the-art CMIP6 models (more info in Carbon Brief website.)

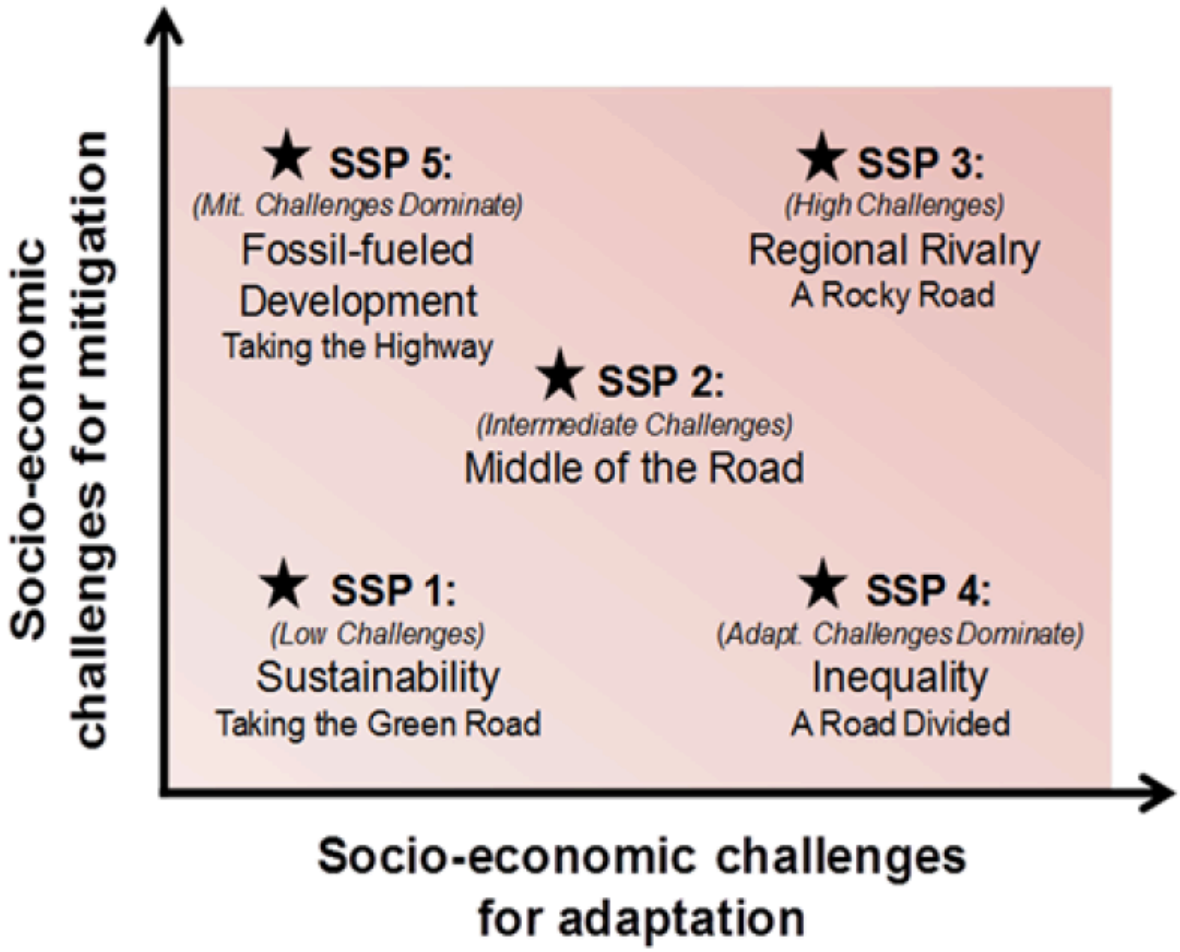
The main goal is to set future climate scenarios based on *future* concentrations of **greenhouse gases**, **aerosols** and other **climate forcings** to project what might happen in the future.



Schematic of the CMIP/CMIP6 experimental design

2.1 Most common climate scenarios

From CMIP5 version you will find those as RCPs (Representative Concentration Pathways) but for the new CMIP6 version they are called SSPs (Shared Socio-Economic Pathways).



Overview of SSPs

Most commonly SSPs used:

- SSP1-2.6: An optimistic scenario, characterised by a shift to a more sustainable economy and reduction in inequality resulting in a peak in radiative forcing of ~3 W m-2 before 2100
- SSP2-4.5: An intermediate scenario, with a stabilisation of radiative forcing levels at ~4.5 W m-2 by 2100
- SSP5-8.5: Characterised by a continued increase of greenhouse gas emissions resulting from a fossil-fuel-based economy and increased energy demand, with a radiative forcing >8.5 W m-2 by 2100

2.2 How to download CMIP6 models

CMIP6 models are free available at the **Earth System Grid Federation** website. You will need an account to download models. **Check this tutorial** of how to create an account.

Through the website:

- Click here to open the ESGF website
- Go to the *Nodes* tab to explore the different ESGF-CoG nodes
- Click the NCI link, the Australia National Computational Infrastructure node

Select a NCI node, go to collection and then CMIP6 link. This is the main website to download CMIP6 models. At your left you have several filters that you can play with, my advice is filter first for variable.

2.2.1 Variables

There are a range of variables available from the GCM (General Circulation Model) outputs. Each tab has different model variable on different time scales. The tabs are in alphabetical order. The ones starting with "O" are for Ocean, and then followed by the timescale (clim = climatology, day, dec = decade, mon = month, yr) (source: Mathematical Marine Ecology Welcome Book Chapter 9).

From the left tab:

- click the + in the **variable** option. Select **tos** (ocean temperature on surface). Then click **Search**
- click the + in the **Realm** option. Select **ocean** and **ocnBgChem**. Then click **Search**
- click the + in the **Frequency** option. Select **mon**. Then click **Search**
- click the + in the **Variant Label** option. Select **r1i1p1f1** (this is the most common ensemble). Then click **Search**

The ensemble names "r1i1p1", "r2i1p1", etc. in **Variant Label** indicate that the ensemble members differ only in their initial conditions (the model physics are the same for all ensemble members, but the members were initialized from different initial conditions out of the control simulation). Hence, the differences between the ensemble members represent internal variability.

- click the + in the **Experiment ID** option. In this option you will see every single *Experiment/Simulation*. For example, **G1/G6/G7** are the geoengineering climate scenarios. Go to the bottom of the **Experiment ID** tab and click **ssp126**. Then click **Search**
- click the + in the **Source ID** option for the full model list and their Institution ID. Let's click on **ACCESS-CM2** model from the CSIRO. Then click **Search**

If you have follow the previous steps you should get a tab result similar to this:

To download the model, just click on the **List Files** tab and then select **HTTP Download**

Chapter 3

Getting Started with Climate Data Operators (CDO)

CMIP6 models come in netCDF file format and they are usually really messy to work in R. For example, the resolution of the **CMIP6** NOAA model (SSP1-2.6) is 0.25°. Let say that you want to download that model for the **thetao** variable (sea temperature with depth). The size of that model is ~80gb. It will be really hard just to read that file in R.

The intention with this information and scripts is to provide a basic understanding of how you can use **CDO** to speed-up your **netCDF** file data manipulation. More info go directly to the Max Planck Institute CDO website

3.1 Installation Process

3.1.1 MacOS

Follow the instruction and downloaded **MacPorts**. **MacPorts** is an open-source community initiative to design an easy-to-use system for compiling, installing, and upgrading the command-line on the Mac operating system.

MacPorts website MacPorts download

After the installation (if you have admin rights) open the terminal and type:

```
port install cdo
```

If you don't have admin rights, open the terminal and type:

`sudo port install cdo` and write your password

3.1.2 Windows 10

In the current windows 10 version(s) Microsoft includes an Ubuntu 16.04 LTS embedded Linux. This environment offers a clean integration with the windows file systems and and the opportunity to install CDO via the native package manager of Ubuntu.

Install the Ubuntu app from the Microsoft Store application. Then open the Ubuntu terminal and type:

`sudo apt-get install cdo` and write your password

3.2 Linux

For Linux go to: Linux

3.3 Ncview: a netCDF visual browser

Ncview is quick visual browser that allows you to explore **netCDF** files very easily: `ncview`. `ncview` is an easy to use netCDF file viewer for **linux** and **OS X**. It can read any netCDF file.

To install **ncview**, open the terminal and type:

- **OS X**: `port install ncview`
- **Linux**: `sudo apt-get install ncview`

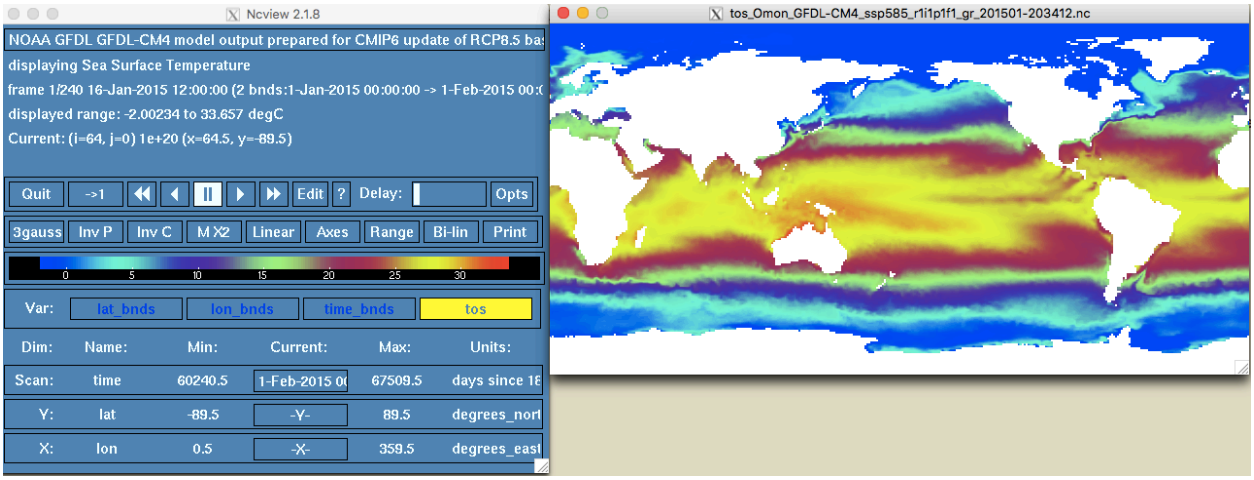
3.4 Working with CDO

To work with **CDO** and **ncview** we need to use the terminal command line: the **Ubuntu app** in Windows and the **Terminal** on OS X.

```
# Establish the primary directory (OS or Linux)
## cd ~/data/ClimateModels/tos/ssp585/

# Establish the primary directory in Windows. It should be located at "/mnt/c/"
## cd ~/data/ClimateModels/tos/ssp585/

# View the model with ncview
## ncview tos_Omon_GFDL-CM4_ssp585_r1i1p1f1_gr_201501-203412.nc
```



ncview example

```
# Check the details by typing in the terminal
## cdo -sinfov tos_Omon_GFDL-CM4_ssp585_r1i1p1f1_gr_201501-203412.nc
```

```
File format : NetCDF4 classic zip
 -1 : Institut Source   T Steptype Levels Num    Points Num Dtype : Parameter name
  1 : unknown  GFDL-CM4 v instant       1   1     64800   1  F32z : tos
Grid coordinates :
  1 : lonlat                  : points=64800 (360x180)
                        lon : 0.5 to 359.5 by 1 degrees_east  circular
                        lat : -89.5 to 89.5 by 1 degrees_north
                  available : cellbounds
Vertical coordinates :
  1 : surface                 : levels=1
Time coordinate :  240 steps
```

Model details

3.4.1 Regrid process

To regrid a **netCDF** file using **CDO** we need to use the argument `remapbil`, which is the stands for bilinear interpolation (there are other methods but this is the most conservative approach). CDO Syntax works like this:

```
# Type in the terminal
## cdo remapbil,r360x180 tos_Omon_GFDL-CM4_ssp585_r1i1p1f1_gr_201501-203412.nc test.nc
```

The previous line will create an uniform file at 1deg of spatial resolution. It works fine for one or two **netCDF** files. However, for multiple **netCDF** files/models (which is most cases of CMIP6 models) the best way to auto the process is to write an *R* script that calls CDO through the **system()** function.

This function will help to search for ANY **netCDF** file in a particular directory and do the "regridding process".

- **Key CDO function**: `remapbil`

```
# This code was written by Isaac Brito-Morales (i.britomorales@uq.edu.au)
# Please do not distribute this code without permission.
# NO GUARANTEES THAT CODE IS CORRECT
# Caveat Emptor!

# Function's arguments
# ipath: directory where the netCDF files are located
# opath: directory to allocate the new regrided netCDF files
# resolution = resolution for the regrid process

regrid <- function(ipath, opath, resolution) {

################################################################################
####### Defining the main packages
################################################################################
  # List of pacakges that we will be used
    list.of.packages <- c("doParallel", "parallel", "stringr", "data.table")
  # If is not installed, install the pacakge
    new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
    if(length(new.packages)) install.packages(new.packages)
  # Load packages
    lapply(list.of.packages, require, character.only = TRUE)

################################################################################
####### Getting the path and directories for the files
```

```
##############################################################################################
  # Establish the find bash command
    line1 <- paste(noquote("find"), noquote(ipath), "-type", "f", "-name",
                   noquote("*.nc"), "-exec", "ls", "-l", "{}")
    line2 <- paste0("\\", ";")
    line3 <- paste(line1, line2)
  # Getting a list of directories for every netCDF file
    dir_files <- system(line3, intern = TRUE)
    dir_nc <- strsplit(x = dir_files, split = " ")
    nc_list <- lapply(dir_nc, function(x){f1 <- tail(x, n = 1)})
  # Cleaning the directories to get a final vector of directories
    final_nc <- lapply(nc_list, function(x) {
      c1 <- str_split(unlist(x), pattern = "//")
      c2 <- paste(c1[[1]][1], c1[[1]][2], sep = "/")})
    files.nc <- unlist(final_nc)

##############################################################################################
####### Starting the regrid process
##############################################################################################
  # Resolution
    if(resolution == "1") {
      grd <- "r360x180"
    } else if(resolution == "0.5") {
      grd <- "r720x360"
    } else if(resolution == "0.25") {
      grd <- "r1440x720"
    }

  # Parallel looop
    UseCores <- 3 # we can change this number
    cl <- makeCluster(UseCores)
    registerDoParallel(cl)
    foreach(j = 1:length(files.nc), .packages = c("stringr")) %dopar% {
      # Trying to auto the name for every model
        var_obj <- system(paste("cdo -showname", files.nc[j]), intern = TRUE)
        var_all <- str_replace_all(string = var_obj, pattern = " ", replacement = "_")
        var <- tail(unlist(strsplit(var_all, split = "_")), n = 1)
      # Running CDO regrid
        system(paste(paste("cdo -remapbil,", grd, ",", sep = ""),
                     paste("-selname",var, sep = ","), files.nc[j],
                     paste0(opath, basename(files.nc[j])), sep = (" "))) # -P 2
    }
    stopCluster(cl)
}
```

Run the regrid R function

```
regrid(ipath = "/data/ClimateModels/",
       opath = "/data/ClimateModelsRegrid/",
       resolution = "0.5")
```

3.4.2 Models with different ocean depths

Some climate models are build across different depths in the ocean. This function will help to search for ANY **netCDF** file in a particular directory, split the file by different ocean depth layer (e.g., surface, epipelagic, mesopelagic, bathypelagic), and merge those files by estimating a vertical average condition.

- **CDO functions**: `showname sellevel selname`
- **Key CDO function**: `vertmean`

```
# This code was written by Isaac Brito-Morales (i.britomorales@uq.edu.au)
# Please do not distribute this code without permission.
# NO GUARANTEES THAT CODE IS CORRECT
# Caveat Emptor!

# Arguments
# ipath: directory where the netCDF files are located
# opath1: directory to allocate the split files
# opath2: directory to allocate the vertical average files

olayer <- function(ipath, opath1, opath2) {

##############################################################################################
####### Defining the main packages (tryining to auto this)
##############################################################################################
  # List of pacakges that we will be used
    list.of.packages <- c("doParallel", "parallel", "stringr", "data.table")
  # If is not installed, install the pacakge
   new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
   if(length(new.packages)) install.packages(new.packages)
  # Load packages
    lapply(list.of.packages, require, character.only = TRUE)

##############################################################################################
####### Getting the path and directories for the files
##############################################################################################
  # Establish the find bash command
    line1 <- paste(noquote("find"), noquote(ipath), "-type", "f", "-name",
                   noquote("*.nc"), "-exec", "ls", "-l", "{}")
    line2 <- paste0("\\", ";")
    line3 <- paste(line1, line2)
  # Getting a list of directories for every netCDF file
    dir_files <- system(line3, intern = TRUE)
    dir_nc <- strsplit(x = dir_files, split = " ")
    nc_list <- lapply(dir_nc, function(x){f1 <- tail(x, n = 1)})
  # Cleaning the directories to get a final vector of directories
    final_nc <- lapply(nc_list, function(x) {
      c1 <- str_split(unlist(x), pattern = "//")
      c2 <- paste(c1[[1]][1], c1[[1]][2], sep = "/")})
    files.nc <- unlist(final_nc)

##############################################################################################
####### Filtering by layers and generating new netCDF files with outputs
##############################################################################################
  # Parallel looop
    cl <- makeCluster(3)
```

```
  registerDoParallel(cl)
  foreach(j = 1:length(files.nc), .packages = c("stringr")) %dopar% {
    # Trying to auto the name for every model
      var_obj <- system(paste("cdo -showname", files.nc[j]), intern = TRUE)
      var_all <- str_replace_all(string = var_obj, pattern = " ", replacement = "_")
      var <- tail(unlist(strsplit(var_all, split = "_")), n = 1)
    # Defining depths
      levels <- as.vector(system(paste("cdo showlevel", files.nc[j]), intern = TRUE))
      lev <- unlist(strsplit(levels, split = " "))
      depths <- unique(lev[lev != ""])
    # Some can come in cm
      if(depths[1] >= 50) {
        sf <- depths[as.numeric(depths) <= 500]
        ep <- depths[as.numeric(depths) >= 0 & as.numeric(depths) <= 20000]
        mp <- depths[as.numeric(depths) > 20000 & as.numeric(depths) <= 100000]
        bap <- depths[as.numeric(depths) > 100000]
      } else {
        sf <- depths[as.numeric(depths) <= 5]
        ep <- depths[as.numeric(depths) >= 0 & as.numeric(depths) <= 200]
        mp <- depths[as.numeric(depths) > 200 & as.numeric(depths) <= 1000]
        bap <- depths[as.numeric(depths) > 1000]
      }
    # Running CDO
      # Surface
        system(paste(paste("cdo -L -sellevel,",
                           paste0(sf, collapse = ","), ",", sep = ""),
                     paste("-selname,", var, sep = ""), files.nc[j],
                     paste0(opath1, "01-sf_", basename(files.nc[j]))))
      # Epipelagic
        system(paste(paste("cdo -L -sellevel,",
                           paste0(ep, collapse = ","), ",", sep = ""),
                     paste("-selname,", var, sep = ""), files.nc[j],
                     paste0(opath1, "02-ep_", basename(files.nc[j]))))
      # Mesopelagic
        system(paste(paste("cdo -L -sellevel,",
                           paste0(mp, collapse = ","), ",", sep = ""),
                     paste("-selname,", var, sep = ""), files.nc[j],
                     paste0(opath1, "03-mp_", basename(files.nc[j]))))
      # Bathypelagic
        system(paste(paste("cdo -L -sellevel,",
                           paste0(bap, collapse = ","), ",", sep = ""),
                     paste("-selname,", var, sep = ""), files.nc[j],
                     paste0(opath1, "04-bap_", basename(files.nc[j]))))
  }
  stopCluster(cl)

##############################################################################
####### Getting the path and directories for the "split by depth" files
##############################################################################
  # Establish the find bash command
    line1.1 <- paste(noquote("find"), noquote(opath1), "-type", "f", "-name",
                     noquote("*.nc"), "-exec", "ls", "-l", "{}")
    line2.1 <- paste0("\\", ";")
```

```
    line3.1 <- paste(line1.1, line2.1)
  # Getting a list of directories for every netCDF file
    dir_files.2 <- system(line3.1, intern = TRUE)
    dir_nc.2 <- strsplit(x = dir_files.2, split = " ")
    nc_list.2 <- lapply(dir_nc.2, function(x){f1 <- tail(x, n = 1)})
  # Cleaning the directories to get a final vector of directories
    final_nc.2 <- lapply(nc_list.2, function(x) {
      c1 <- str_split(unlist(x), pattern = "//")
      c2 <- paste(c1[[1]][1], c1[[1]][2], sep = "/")})
    files.nc.2 <- unlist(final_nc.2)

##############################################################################################
####### Filtering by layers and generating the "weighted-average depth layer"
##############################################################################################
  # Parallel looop
    cl <- makeCluster(3)
    registerDoParallel(cl)
    foreach(i = 1:length(files.nc.2), .packages = c("stringr")) %dopar% {
      # Running CDO
        system(paste(paste("cdo -L vertmean", sep = ""), files.nc.2[i],
                     paste0(opath2, basename(files.nc.2[i])), sep = (" ")))
    }
    stopCluster(cl)
}
```

Running the ocean depth layer R function will:

```
olayer(ipath = "/data/ClimateModelsRegrid/",
       opath1 = "/data/ClimateModelsRegridLayer/",
       opath2 = "/data/ClimateModelsRegridLayerMean/")
```

3.4.3 Merge several netcdf files with CDO

This function will help to merge several ocean depth layers (from the same model) into a single file.

- **Key CDO function**: `mergetime`

```
merge_files <- function(ipath, opath1) {

##############################################################################################
####### Defining the main packages (tryining to auto this)
##############################################################################################
  # List of pacakges that we will be used
    list.of.packages <- c("doParallel", "parallel", "stringr", "data.table")
  # If is not installed, install the pacakge
    new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
    if(length(new.packages)) install.packages(new.packages)
  # Load packages
    lapply(list.of.packages, require, character.only = TRUE)

##############################################################################################
####### Getting the path and directories for the files
##############################################################################################
  # Establish the find bash command
    line1 <- paste(noquote("find"), noquote(ipath), "-type", "f", "-name",
```

```
                  noquote("*.nc"), "-exec", "ls", "-l", "{}")
    line2 <- paste0("\\", ";")
    line3 <- paste(line1, line2)
  # Getting a list of directories for every netCDF file
    dir_files <- system(line3, intern = TRUE)
    dir_nc <- strsplit(x = dir_files, split = " ")
    nc_list <- lapply(dir_nc, function(x){f1 <- tail(x, n = 1)})
  # Cleaning the directories to get a final vector of directories
    final_nc <- lapply(nc_list, function(x) {
      c1 <- str_split(unlist(x), pattern = "//")
      c2 <- paste(c1[[1]][1], c1[[1]][2], sep = "/")})
    files.nc <- unlist(final_nc)

################################################################################
####### Filtering by layers and generating new netCDF files with outputs
################################################################################
  # Filtering (not dplyr!) by ocean layers
    sf <- files.nc[str_detect(string = basename(files.nc), pattern = "01-sf*") == TRUE]
    ep <- files.nc[str_detect(string = basename(files.nc), pattern = "02-ep*") == TRUE]
    mp <- files.nc[str_detect(string = basename(files.nc), pattern = "03-mp*") == TRUE]
    bap <- files.nc[str_detect(string = basename(files.nc), pattern = "04-bap*") == TRUE]
  # Defining how many models are per ocean layer
    model_list_sf <- lapply(sf, function(x)
      {d1 <- unlist(strsplit(x = basename(x), split = "_"))[4]})
    model_list_ep <- lapply(ep, function(x)
      {d1 <- unlist(strsplit(x = basename(x), split = "_"))[4]})
    model_list_mp <- lapply(bap, function(x)
      {d1 <- unlist(strsplit(x = basename(x), split = "_"))[4]})
    model_list_bap <- lapply(bap, function(x)
      {d1 <- unlist(strsplit(x = basename(x), split = "_"))[4]})
    models <- unique(unlist(c(model_list_sf, model_list_ep, model_list_mp, model_list_bap)))
  # Parallel looop
    cl <- makeCluster(3)
    registerDoParallel(cl)
    foreach(i = 1:length(models), .packages = c("stringr")) %dopar% {
      f1 <- ep[str_detect(string = basename(sf), pattern = models[i]) == TRUE]
      system(paste(paste("cdo -L mergetime", paste0(f1, collapse = " "), sep = " "),
                   paste0(opath1, paste(unlist(strsplit(basename(f1[1]), "_"))[c(1:7)],
                                        collapse = "_"), ".nc"), sep = (" ")))
      f2 <- ep[str_detect(string = basename(ep), pattern = models[i]) == TRUE]
      system(paste(paste("cdo -L mergetime", paste0(f2, collapse = " "), sep = " "),
                   paste0(opath1, paste(unlist(strsplit(basename(f2[1]), "_"))[c(1:7)],
                                        collapse = "_"), ".nc"), sep = (" ")))
      f3 <- mp[str_detect(string = basename(mp), pattern = models[i]) == TRUE]
      system(paste(paste("cdo -L mergetime", paste0(f3, collapse = " "), sep = " "),
                   paste0(opath1, paste(unlist(strsplit(basename(f3[1]), "_"))[c(1:7)],
                                        collapse = "_"), ".nc"), sep = (" ")))
      f4 <- bap[str_detect(string = basename(bap), pattern = models[i]) == TRUE]
      system(paste(paste("cdo -L mergetime", paste0(f4, collapse = " "), sep = " "),
                   paste0(opath1, paste(unlist(strsplit(basename(f4[1]), "_"))[c(1:7)],
                                        collapse = "_"), ".nc"), sep = (" ")))
    }
    stopCluster(cl)
```

```
}
```

Running the merge function

```
merge_files(ipath = "/Users/bri273/Desktop/CDO/models_regrid_vertmean/",
            opath1 = "/Users/bri273/Desktop/CDO/models_regrid_zmerge/")
```

The functions above were build based on OS. For Linux please check:

- **regrid**: regrid R function
- **ocean layers**: ocean layer R function
- **merge**: calculates merge R function

3.4.4 Some useful CDO functions

CDO is more than just regridding. Some interesting useful functions are:

- **yearmean**: calculates the **annual mean** of a monthly data input **netCDF** file
- **yearmin**: calculates the **annual min** of a monthly data input **netCDF** file
- **yearmax**: calculates the **annual max** of a monthly data input **netCDF** file
- **ensmean**: calculates the **ensemble mean** of several **netCDF** files. If the input files are different models, this function will estimate a mean of all those models

Chapter 4

netCDF files in R: Raster, Spatial objects

4.1 Introduction

The aim of this tutorial is to provide a worked example (i.e., a function) of how to transform a regridded **netCDF** into a Raster object using R.

4.2 Data import

Load the required packages.

```
# load packages
library(raster)
library(ncdf4)
library(ncdf4.helpers)
library(PCICt)
```

4.3 Function to transform netCDF files into Raster objects

You can read netCDF using `raster:::stack` or the `raster:::terra` functions from the `raster` and `terra` packages. However, this function allows more control over the outputs.

```
# This code was written by Isaac Brito-Morales (i.britomorales@uq.edu.au)
# Please do not distribute this code without permission.
# NO GUARANTEES THAT CODE IS CORRECT
# Caveat Emptor!

  ncdf_2D_rs <- function(nc, from, to, v = "tos", x = "lon", y = "lat") {
  # Extract data from the netCDF file
   nc <- nc_open(nc)
   dat <- ncvar_get(nc, v) # x, y, year
   dat[] <- dat
   X <- dim(dat)[1]
   Y <- dim(dat)[2]
   tt <- nc.get.time.series(nc, v = "time", time.dim.name = "time")
   tt <- as.POSIXct(tt)
```

```
 tt <- as.Date(tt)
 nc_close(nc)
 rs <- raster(nrow = Y, ncol = X) # Make a raster with the right dims
# Fix orientation of original data
  drs <- data.frame(coordinates(rs))
# Create Rasters Stack
  rs_list <- list() # empty list to allocate results
  st <- stack()
  for (i in 1:length(tt)) {
    dt1 <- rasterFromXYZ(cbind(drs, as.vector(dat[,, i])))
    dt1[]<- ifelse(dt1[] <= -2, NA, dt1[]) # for some models that have weird temperatures
    dt1[]<- ifelse(dt1[] >= 40, NA, dt1[]) # for some models that have weird temperatures
    st <- addLayer(st, flip(dt1, 2))
    print(paste0(i, " of ", length(tt)))
  }
  names(st) <- seq(as.Date(paste(from, "1", "1", sep = "/")),
                   as.Date(paste(to, "12", "1", sep = "/")), by = "month")
  crs(st) <- "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
  return(st)
  }
```

Chapter 5

Climate Change Metrics

5.1 Introduction

The aim of this tutorial is to provide a worked example of how to calculate climate change metrics using CMIP6 models. The climate-change metrics used in this example are **Climate Velocity** and a **Relative Climate Exposure** index.

This dataset contains a raster-stack file in format **.grd** of monthly sea surface temperature (tos) for the GFDL-CM4 model under a SSP5-5.8 emission scenario. The model goes from 2015 until 2100.

5.2 Data import

Load the required packages and the data.

```
# load packages
  library(raster)
  library(VoCC)
  library(stringr)
  library(dplyr)
```

5.3 Climate Velocity (VoCC)

Climate velocity is a vector that describes the speed and direction that a point on a gridded map would need to move to remain static in climate space (e.g., to maintain an isoline of a given variable in a univariate environment) under climate change. From an ecological perspective, *climate velocity* can be conceptualized as the speed and direction in which a species would need to move to maintain its current climate conditions under climate change. For this reason, *climate velocity* can be considered to represent the potential exposure to climate change faced by a species if the climate moves beyond the physiological tolerance of a local population. See Brito-Morales et al. 2018.

Install the R package: GitHub Repo

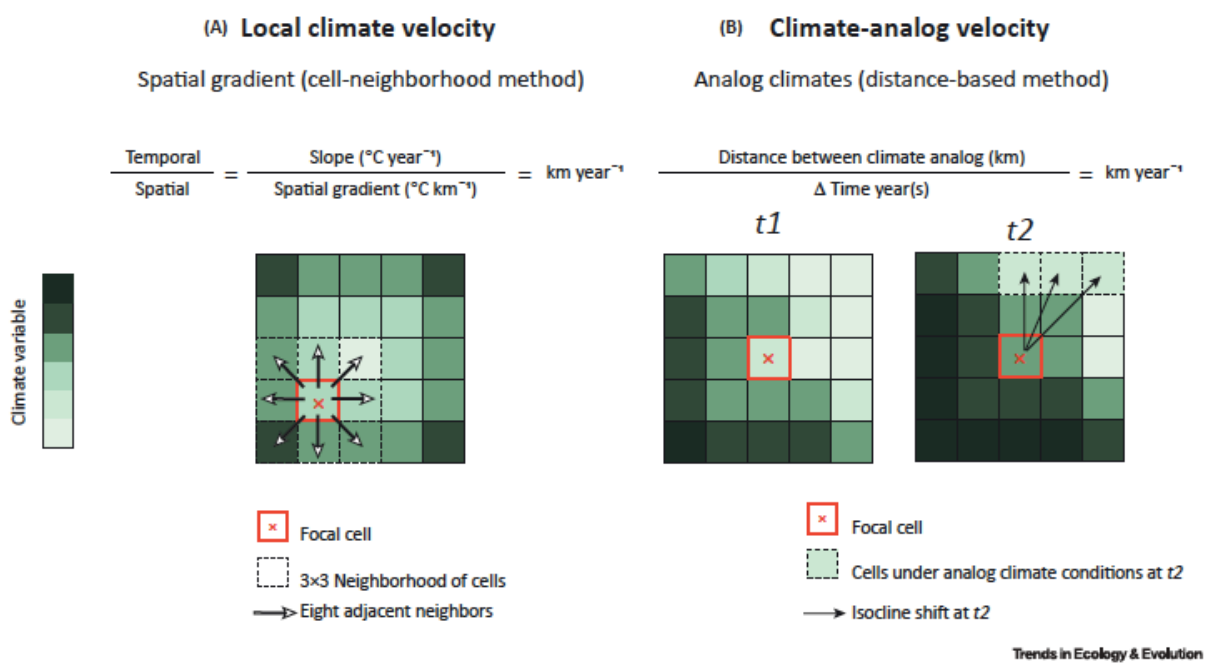
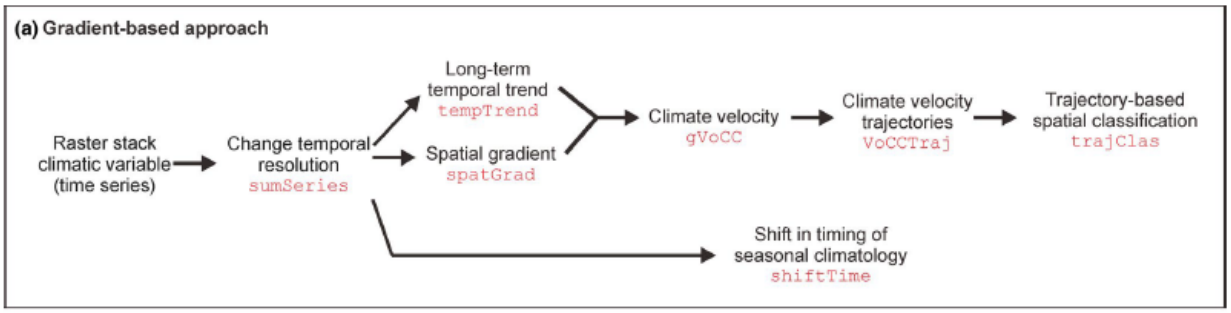


Figure 1. Mathematical and Graphical Differences between (A) Local Climate and (B) Climate-Analog Velocities.

To calculate *climate velocity* the R package `VoCC` provides a comprehensive collection of functions that calculate climate velocity and related metrics from their initial formulation to the latest developments. See Garcia Molinos et al. 2019.




```
# Load the monthly raster object
  rs <- raster::stack("data/ClimateModelsRasterMonthly/tos/ssp585/tos_Omon_GFDL-CM4_ssp585_2015-2100.gr
# Establish the time period of interest (if applicable)
  from = 2020
  to = 2100
# Define the time period to estimate climate velocity and subset the original raster
  names.yrs <- paste("X", seq(as.Date(paste(from, "1", "1", sep = "/")), as.Date(paste(to, "12", "1", s
    str_replace_all(pattern = "-", replacement = ".")
  rs <- raster::subset(rs, names.yrs)
# If Raster is monthly, get annual mean
  index <- rep(1:nlayers(rs), each = 12, length.out = nlayers(rs))
  rs <- stackApply(x = rs, indices = index, fun = mean)

# Calculate VoCC
  # Temporal trend (slope)
    slp <- tempTrend(rs, th = 10)
```

```
  # Spatial gradient (gradient)
    grad <- spatGrad(rs, th = 0.0001, projected = FALSE)
  # VoCC local gradient
    vocc <- gVoCC(slp, grad)
    vocc$voccMag[] <- ifelse(is.infinite(vocc$voccMag[]), NA, vocc$voccMag[]) # replace inf with NAs
```

5.4 Relative Climate Exposure (RCE)

The RCE is a metric that we developed to obtain information about the amount of exposure to climate warming that local populations of a species would face relative to its experience of variation in seasonal temperatures Brito-Morales et al. 2021. RCE is calculated as the ratio of the slope of a linear regression of projected mean annual temperatures (°C yr-1) to the current mean seasonal temperature range (°C):

```
# The current seasonal variation (could be any range)
  from = 2016
  to = 2021
# Getting the years/month to calculate de RCE index
  names.yrs <- paste("X", seq(as.Date(paste(from, "1", "1", sep = "/")),
                              as.Date(paste(to, "12", "1", sep = "/")), by = "month"),
                     sep = "") %>%
    str_replace_all(pattern = "-", replacement = ".")
# Read and subset the data
  rs <- readAll(raster::stack("data/ClimateModelsRasterMonthly/tos/ssp585/tos_Omon_GFDL-CM4_ssp585_2015-
    subset(names.yrs)
# Annual min and max to estimate the rage to get the RCE
  index <- rep(1:nlayers(rs), each = 12, length.out = nlayers(rs))
  rs_min <- stackApply(x = rs, indices = index, fun = min)
  rs_max <- stackApply(x = rs, indices = index, fun = max)
# Range among the period selected
  rs_range <- rs_max - rs_min
  rs_range_mean <- stackApply(x = rs_range, indices = nlayers(rs_range), fun = mean)

# Get the slope
  from = 2020
  to = 2100
  names.yrs <- paste("X", seq(as.Date(paste(from, "1", "1", sep = "/")), as.Date(paste(to, "12", "1", s
    str_replace_all(pattern = "-", replacement = ".")
  rs <- raster::stack("data/ClimateModelsRasterMonthly/tos/ssp585/tos_Omon_GFDL-CM4_ssp585_2015-2100.gr
    subset(names.yrs)
  index <- rep(1:nlayers(rs), each = 12, length.out = nlayers(rs))
  rs <- stackApply(x = rs, indices = index, fun = mean)
  slp <- ((tempTrend(rs, th = 10))*10)*8 # x10 decadal and x8 for decades (2020-2100)

# Calculate the RCE
  RCE <- abs(slp/rs1_range_mean)
```

Chapter 6

Raster objects and Spatial object

6.1 Introduction

The aim here is to provide a worked example of how intersect rasters with spatial polygon objects using the *sf R* and *exactextractr R* packages.

This example used two objects.

- **spatial polygon object**: located at `data/PlanningUnits/PUs_WeddellSea_100km2.shp`. This object contains hexagonal polygons of equal area. It has two columns: integer unique identifiers ("id"), geometry information ("geometry")

- **raster object**: located at `data/VoCC/tos/voccMag_tos_GFDL-CM4_ssp585_2015-2100.grd_2050-2100_.tif`. This object is a raster *.tif* object of climate-velocity estimates in the southern ocean.

The objective here is to assign to eash hexagon in the *spatial polygon object* a climate velocity value

6.2 Data import

First, load the required packages and the data.

```
# load packages
  library(raster)
  library(sf)
  library(dplyr)
  library(exactextractr)
  library(stringr)
  library(nngeo)
```

6.3 Function to replace NAs with nearest neighbor

This functions helps to replace NAs with nearest neighbor interpolation method.

```
# Function to replace NAs with nearest neighbor. Function wrtitten by Jason Everett
  fCheckNAs <- function(df, vari) {
    if (sum(is.na(pull(df, !!sym(vari))))>0){ # Check if there are NAs
      gp <- df %>%
        mutate(isna = is.finite(!!sym(vari))) %>%
        group_by(isna) %>%
```

```
      group_split()

    out_na <- gp[[1]] # DF with NAs
    out_finite <- gp[[2]] # DF without NAs

    d <- st_nn(out_na, out_finite) %>% # Get nearest neighbour
      unlist()
    out_na <- out_na %>%
      mutate(!!sym(vari) := pull(out_finite, !!sym(vari))[d])
    df <- rbind(out_finite, out_na)
  }
  return(df)
}
```

6.4 Raster by spatial polygon object

The aim here is to provide an example of how integrate raster in an sf polygon spatial object

```
# Reading the spatial polygon object
pu_region <- st_read("data/PlanningUnits/PUs_WeddellSea_100km2.shp")
st_crs(pu_region) <- "+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs"

# Reading the raster object
vocc_file <- raster::raster("data/VoCC/tos/voccMag_tos_GFDL-CM4_ssp585_2015-2100.grd_2050-2100_.tif")
crs(vocc_file) <- CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0")
weight_rs_vocc <- raster::area(vocc_file)
vocc_file <- projectRaster(vocc_file, crs = CRS("+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=W
                           method = "ngb", over = FALSE)
weight_vocc <- projectRaster(weight_rs_vocc, crs = CRS("+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +
                             method = "ngb", over = FALSE)
names(vocc_file) <- "layer"

# Getting the value by planning unit
vocc_bypu <- exact_extract(vocc_file, pu_region, "weighted_mean", weights = weight_vocc)
vocc_shp <- pu_region %>%
  dplyr::mutate(vocc = vocc_bypu) %>%
  dplyr::relocate(cellID, vocc)
# Replace NAs with nearest neighbor
vocc_sfInt <- fCheckNAs(df = vocc_shp, vari = names(vocc_shp)[2]) %>%
dplyr::select(-isna)
```

6.5 Spatial polygon object by region of interest

The aim here is to provide a simple script to assign a Longhurst province identifier per planning unit

```
# Reading the spatial polygon object
pu_region <- st_read("data/PlanningUnits/PUs_WeddellSea_100km2.shp") %>%
  st_transform(crs = CRS("+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs
# Reading Longhurst Provinces Shapefile
bioprovince <- st_read("data/Boundaries/LonghurstProvinces/Longhurst_world_v4_2010.shp") %>%
  st_transform(crs = CRS("+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs
  st_make_valid()
# Get the Longhurst Provinces per Planning unit
```

```
nr <- st_nearest_feature(pu_region, bioprovince)
pu_region <- pu_region %>%
  dplyr::mutate(province = paste(as.vector(bioprovince$ProvCode[nr]), prov_name, sep = "_")) %>%
  dplyr::arrange(layer)
```