Ocean Fronts Change Workshop 2023

Isaac Brito-Morales

2023-07-22

Table of contents

Preface 4

Acknowledgements 5

1 Welcome! 6

2 Setting up your computer 8
2.1 R . . . 8
2.2 RStudio . . . 8
2.3 R packages . . . 9

3 Getting Started with Front/Climate Data 11
3.1 Installation Process . . . 11
3.1.1 **MacOS** . . . 11
3.1.2 **Windows 10** . . . 11
3.1.3 **Linux** . . . 11
3.2 Ncview: a netCDF visual browser . . . 12

4 netCDF files in R: Raster, Spatial objects 13
4.1 Data import . . . 13
4.2 Function to transform netCDF files into Raster objects . . . 13

5 Equal-area grid for your area of interest. 15
5.1 Data import . . . 15
5.2 Polygons on the area of interest . . . 15
5.3 Planning units for the whole region . . . 15
5.4 Final equal-area grid . . . 16
5.5 Plot the output . . . 16

6 Fronts (or any variable) by equal-size grid 17
6.1 Data import . . . 17
6.2 Generic Function to replace NAs with nearest neighbor. . . . 17
6.3 Read, Extract and weighted mean interpolation . . . 18
6.4 RUN: replace NAs with nearest neighbor. . . . 18
6.5 Plot the output . . . 19

7 Megafauna and Ocean Fronts **20**
7.1 Data import 20
7.2 Plotting and testing 21
7.3 Near distance to Fronts 24
7.4 Clean (yes, even more!) 26
7.5 Distance Histograms 27

8 Appendix 1: Random Walk analysis in R **29**
8.1 Data import 29

References **30**

Preface

All the code was written by Isaac Brito-Morales (ibrito@conservation.org)

Please do not distribute this code without permission

NO GUARANTEES THAT CODE IS CORRECT

Caveat Emptor!

Acknowledgements

Ocean Front data: Floriande Sudre, Mediterranean Institute of Oceanography, France

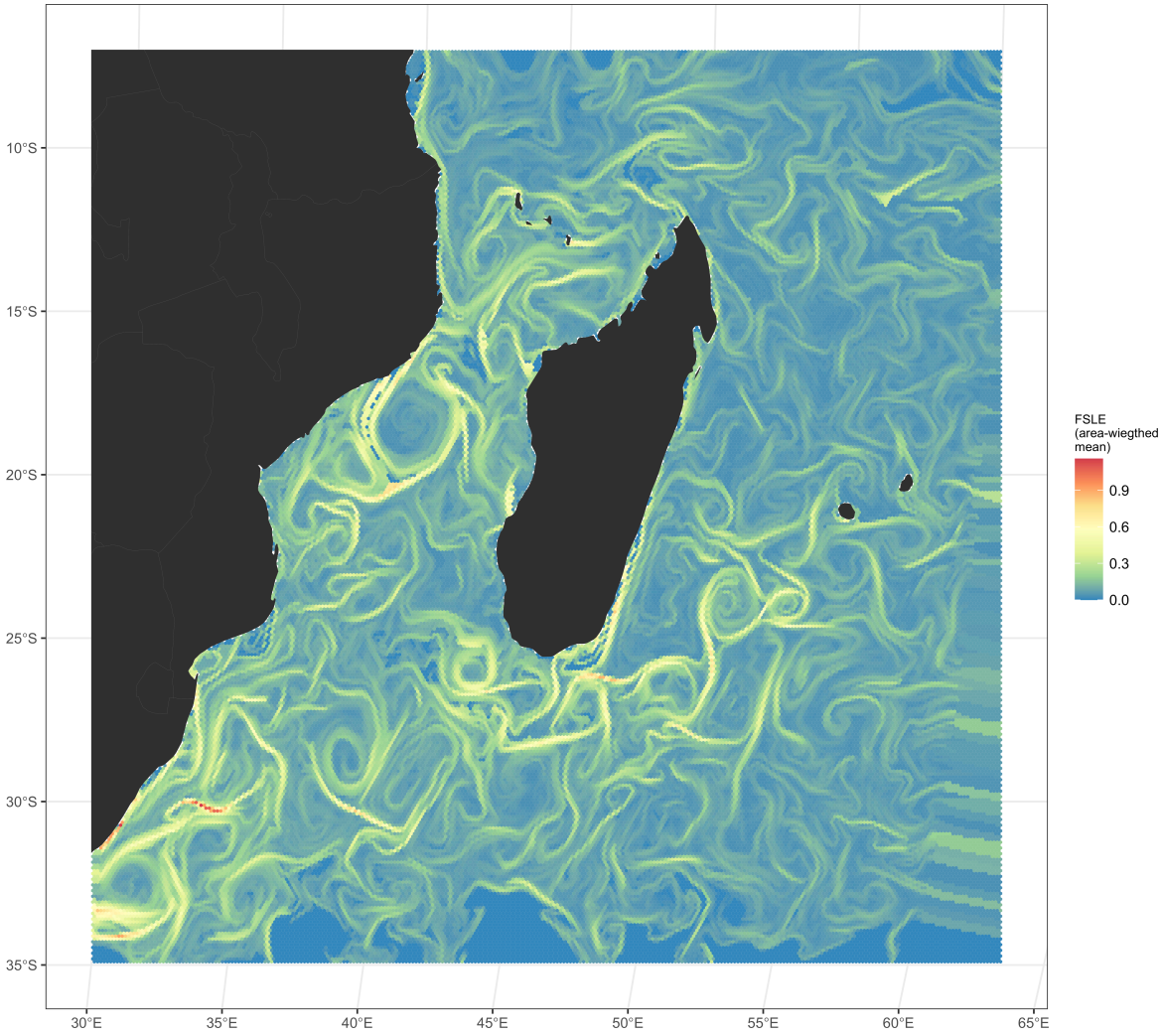
Whale shark dataset: Dr Chris Ronner, Marine Megafauna Foundation

1 Welcome!

Welcome to the Ocean Fronts Change Megafauna Workshop! This workshop is specifically designed to assist you in getting started with ocean fronts and megafauna data analysis using R.

Please note that the objective of this workshop is not to provide a comprehensive overview that would make you an expert in this field. Rather, our goal is to help you grasp the core principles underlying the relationships between ocean fronts and megafauna, and to guide you through some of the common tasks involved in analyzing and interpreting such data.

The workshop will run 9-3 pm (South Africa Time) on Tuesday 25th, Wednesday 26th and Thursday 27 July 2021. The rough plan for the workshop is to cover Chapters 1-3 on Day 1, and Chapters 4-5 on Day 2. We hope this will leave lots of time for discussion.


10°S
15°S
20°S
25°S
30°S
35°S
30°E
35°E
40°E
45°E
50°E
55°E
60°E
65°E
FSLE
(area-wiegthed
mean)
0.9
0.6
0.3
0.0

2 Setting up your computer

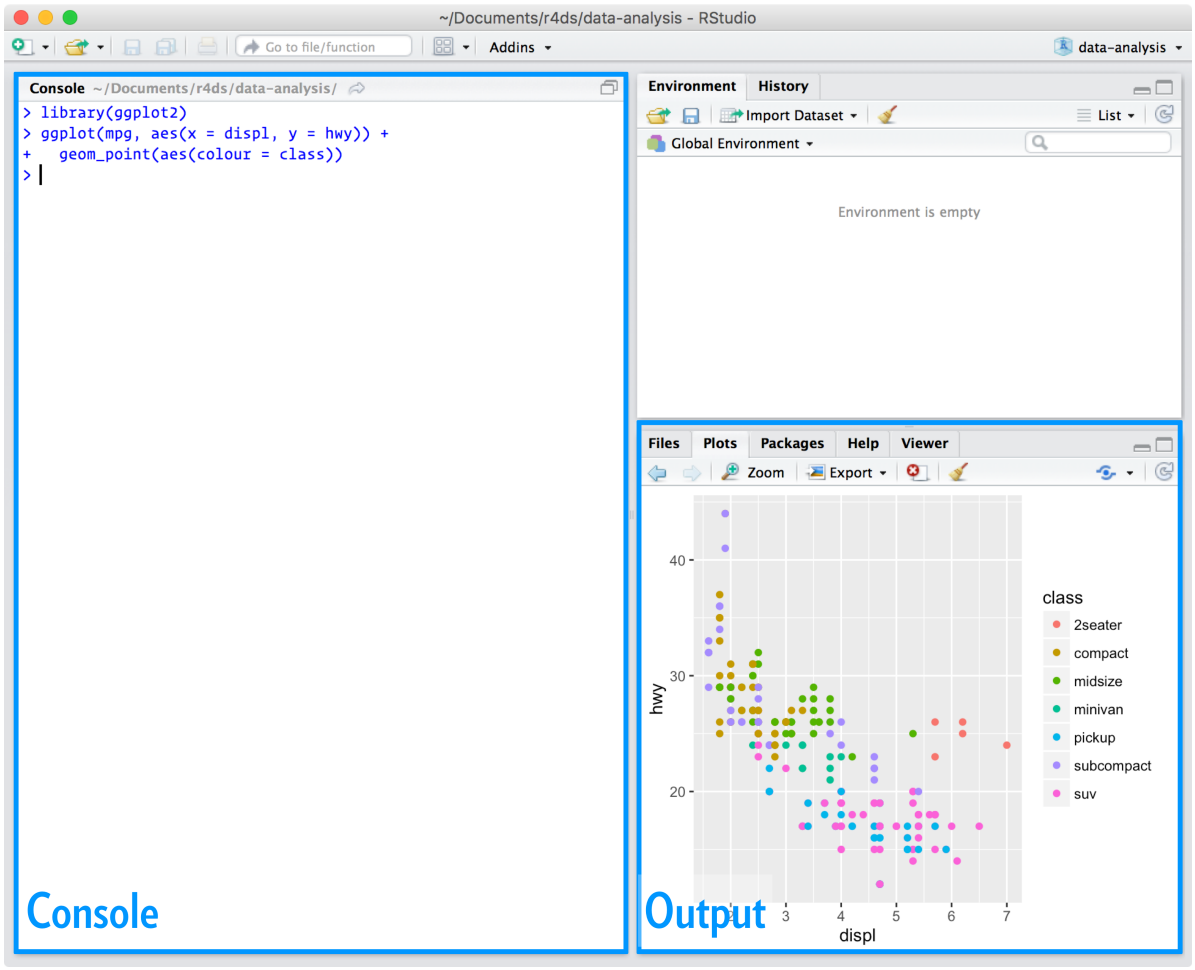
You will need to have both R and RStudio installed on your computer to complete this workshop. Although it is not imperative that you have the latest version of RStudio installed, **you will need to have at least version 4.0 of R installed** . Please note that you might need administrative permissions to install these programs. After installing them, you will also need to install some R packages too. Finally, you will also need to download the data for this workshop.

2.1 R

The R statistical computing environment can be downloaded from the Comprehensive R Archive Network (CRAN). Specifically, you can download the latest version of R (version 4.2.3) from here: https://cloud.r-project.org. Please note that you will need to download the correct file for your operating system (i.e. Linux, Mac OSX, Windows).

2.2 RStudio

RStudio is an integrated development environment (IDE). In other words, it is a program that is designed to make your R programming experience more enjoyable. During this workshop, you will interact with R through RStudio—meaning that you will open RStudio to code in R. You can download the latest version of RStudio here: http://www.rstudio.com/download. When you start RStudio, you will see two main parts of the interface:



You can type R code into the *Console* and press the enter key to run code.

2.3 R packages

An R package is a collection of R code and documentation that can be installed to enhance the standard R environment with additional functionality. Currently, there are over fifteen thousand R packages available on CRAN. Each of these R packages are developed to perform a specific task, such as reading Excel spreadsheets, downloading satellite imagery data, downloading and cleaning protected area data, or fitting environmental niche models. In fact, R has such a diverse ecosystem of R packages, that the question is almost always not "can I use R to ...?" but "what R package can I use to ...?". During this workshop, we will use several R packages. To install these R packages, please enter the code below in the *Console* part of the RStudio interface and press enter. Note that you will require an Internet connection and the installation process may take some time to complete.

```r
install.packages(c("sf", "terra", "dplyr", "sp", "rgeos", "rgdal", "raster",
                   "units", "tidyr", "stringr", "readr", "transformr", "data.table",
                   "ggplot2", "RColorBrewer", "patchwork", "rnaturalearth", "rnaturalearth
                   "ggtext", "lwgeom"))

# library(gganimate)
# library(animation)
# library(ncdf4)
# library(ncdf4.helpers)
# library(PCICt)
# library(magrittr)
# library(exactextractr)
# library(nngeo)
# library(stringr)
```

3 Getting Started with Front/Climate Data

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

3.1.3 Linux

For Linux go to: Linux

3.2 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`

4 netCDF files in R: Raster, Spatial objects

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.

4.1 Data import

Load the required packages.

```
# load packages
library(terra)
library(ncdf4)
library(ncdf4.helpers)
library(PCICt)
library(dplyr)
library(magrittr)
```

4.2 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 code allows more control over the outputs.

```
# NO GUARANTEES THAT CODE IS CORRECT
# Caveat Emptor!

  nc = "data/BOAonMUR_SWIO_Y2003-M1-D1.nc"
  v = "temp_gradient"
  x = "lon"
  y = "lat"

  nc <- ncdf4::nc_open(nc)
  dat <- ncdf4::ncvar_get(nc, v) # x, y, year
  dat[] <- dat
```

```
  rlon <- ncdf4::ncvar_get(nc, varid = x) %>%
    range()
  rlat <- ncdf4::ncvar_get(nc, varid = y) %>%
    range()
  X <- dim(dat)[1] # nolint
  Y <- dim(dat)[2] # nolint
  tt <- ncdf4.helpers::nc.get.time.series(nc,
                                          v = "time",
                                          time.dim.name = "time")
  tt <- as.POSIXct(tt)
  tt <- as.Date(tt)
  ncdf4::nc_close(nc)
  # Make a raster with the right dims to fill with lat&lon
    rs <- terra::rast(nrow = Y, ncol = X, extent = terra::ext(c(rlon, rlat)))
    # Fix orientation of original data
    # [and then create a raster with this fix orientation]
    drs <- terra::xyFromCell(rs, 1:terra::ncell(rs)) %>%
      as_tibble()
    # Create raster stacks of depths for every month
    rs_list <- list() # to allocate results # nolint
    st <- terra::rast()
    for (i in 1:length(tt)) { # nolint
      dt1 <- dplyr::bind_cols(drs,
                              as.vector(dat[i])) %>%
        magrittr::set_colnames(c("x", "y", v))
      dt1 <- terra::rast(dt1, type = "xyz")
      names(dt1) <- tt[i]
      st <- c(st, terra::flip(dt1))
      print(paste0(tt[i], " of ", length(tt)))
    }
```

Crop/Manipulate and project

```
# NO GUARANTEES THAT CODE IS CORRECT
# Caveat Emptor!
  st <- terra::crop(st, terra::ext(c(30, 75, -35, -7)))
  terra::crs(st) <- "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

5 Equal-area grid for your area of interest.

5.1 Data import

Load the required packages.

```
source("zscripts/z_helpFX.R")
```

5.2 Polygons on the area of interest

```
  bbox <- st_bbox(c(xmin = 30, xmax = 60, ymax = -7, ymin = -35),
                  crs = st_crs("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")) %>%
    st_as_sfc() %>%
    st_transform(crs = robin)

# taking 0.1 from borders to avoid border effect
  f_bbox <- st_bbox(c(xmin = 30.1, xmax = 59.9, ymax = -7.1, ymin = -34.9),
                    crs = st_crs("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")) %>%
    st_as_sfc() %>%
    st_transform(crs = robin)
# Area
  CellArea <- 100 # in km2
  h_diameter <- 2 * sqrt((CellArea*1e6)/((3*sqrt(3)/2))) * sqrt(3)/2 # Diameter in m
  s_diameter <- sqrt(CellArea*1e6) # Diameter in m
```

5.3 Planning units for the whole region

```
  PUs <- st_make_grid(f_bbox,
                      square = F,
                      cellsize = c(h_diameter, h_diameter),
                      what = "polygons",
```

```
                      crs = st_crs(f_bbox)) %>%
    st_sf()

# Check cell size worked OK.
  print(paste0("Range of cellsize are ",
               round(as.numeric(range(units::set_units(st_area(PUs), "km^2")))[1]),," km2 t
               round(as.numeric(range(units::set_units(st_area(PUs), "km^2")))[2]),," km2")
```

5.4 Final equal-area grid

```
  logi_PUs <- st_centroid(PUs) %>%
    st_intersects(world_sfRob) %>%
    lengths > 0 # Get logical vector instead of sparse geometry binary
  PUs1 <- PUs[logi_PUs == FALSE, ]
  plot(st_geometry(PUs1))
```

5.5 Plot the output

```
  g1 <- ggplot() +
    geom_sf(data = PUs, size = 0.05) +
    geom_sf(data = world_sfRob, size = 0.05, fill = "grey20") +
    theme_bw()

  print(g1)

  # ggsave("MYFILE.png", plot = g1, width = 30, height = 30, dpi = 600, limitsize = FALSE)
  # st_write(obj = PUs, dsn = "MYDIRECTORY", layer = "MYFILE", driver = "ESRI Shapefile")
```

6 Fronts (or any variable) by equal-size grid

6.1 Data import

Load the required packages.

```
library(terra)
library(sf)
library(exactextractr)
library(dplyr)
library(nngeo)
library(stringr)
```

6.2 Generic Function to replace NAs with nearest neighbor.

```
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)
  }
```

6.3 Read, Extract and weighted mean interpolation

```
proj.geo = "+proj=robin +lon_0=0 +datum=WGS84 +units=m +no_defs"
# Reading equal grid file
  shp_file <- st_read("data/PUs_MZ_100km2.shp") %>%
    st_transform(crs = terra::crs(proj.geo))
# Read raster object
  rs_file <- rast("data/BOAonMUR_SWIO_Y2003-M1-D1.nc")
  crs(rs_file) <- terra::crs("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0")
  weight_rs <- terra::cellSize(rs_file)
  rs_file <- terra::project(rs_file, y = terra::crs(proj.geo), method = "near")
  weight_rs <- terra::project(weight_rs, y = terra::crs(proj.geo), method = "near")
# Getting value by polygon
  rs_bypu <- exact_extract(rs_file,
                           shp_file,
                           "weighted_mean",
                           weights = weight_rs,
                           append_cols = TRUE,
                           full_colnames = TRUE)
  rs_shp <- dplyr::right_join(shp_file, rs_bypu, "FID")
  colnames(rs_shp) <- c(stringr::str_remove_all(string = names(rs_shp), pattern = "weighte
```

6.4 RUN: replace NAs with nearest neighbor.

```
nms <- names(rs_shp)
nms <- nms[nms != "geometry" & nms != "FID"]

single <- rs_shp %>%
  dplyr::select(FID, nms[1])
rs_sfInt <- fCheckNAs(df = single, vari = names(single)[2]) %>%
  as_tibble() %>%
  dplyr::arrange(FID) %>%
```

```
  dplyr::select(-FID, -geometry, -isna)

saveRDS(rs_sfInt, "data/BOAonMUR_SWIO_Y2003-M1-D1.rds")
```

6.5 Plot the output

```
pus <- st_read("data/PUs_MZ_100km2.shp") %>%
  st_transform(crs = robin)
sf1 <- readRDS("data/BOAonMUR_SWIO_Y2003-M1-D1.rds")
df1 <- cbind(pus, sf1) %>%
  st_transform(crs = robin)

p1 <- ggplot() +
  geom_sf(data = df1, aes(fill = temp_gradient.area), colour = NA) +
  geom_sf(data = world_sfRob, size = 0.05, fill = "grey20") +
  theme_bw() +
  theme(legend.title = element_text(angle = 0, size = rel(0.7)),
        plot.title = element_text(face = "plain", size = 22, hjust = 0.5),
        axis.text.x = element_text(size = rel(1), angle = 0),
        axis.text.y = element_text(size = rel(1), angle = 0),
        axis.title = element_blank()) +
  scale_fill_distiller(palette = "Spectral",
                       limits = c(min(df1$temp_gradient.area), max(df1$temp_gradient.area)
                       direction = -1,
                       oob = scales::squish,
                       guide = guide_colourbar(title.position = "top", title = "thermal gr
```

7 Megafauna and Ocean Fronts

7.1 Data import

Source the required dataset.

```
source("zscripts/z_inputFls.R")
source("zscripts/z_helpFX.R") # and the Help function just in case :-)
```

Or you can actually do the step-by-step.

```
##############################
####### Whale Shark (mmf dataset)
##############################

# Read the files
  ws01 <- readRDS("data/mm_mmf/MMF_WhaleSharkTracks_Madagascar.rds")
  ws02 <- readRDS("data/mm_mmf/MMF_WhaleSharkTracks_Mozambique.rds")
# Merge
  wsF <- rbind(ws01, ws02) %>%
    dplyr::mutate(group = "Whale Shark")
# Monthly split
  wsF01 <- wsF %>%
    dplyr::filter(dates %in% wsF$dates[stringr::str_detect(string = wsF$dates, pattern = "
  wsF02 <- wsF %>%
    dplyr::filter(dates %in% wsF$dates[stringr::str_detect(string = wsF$dates, pattern = "
  wsF03 <- wsF %>%
    dplyr::filter(dates %in% wsF$dates[stringr::str_detect(string = wsF$dates, pattern = "

  wsF04 <- wsF %>%
    dplyr::filter(dates %in% wsF$dates[stringr::str_detect(string = wsF$dates, pattern = "
  wsF05 <- wsF %>%
    dplyr::filter(dates %in% wsF$dates[stringr::str_detect(string = wsF$dates, pattern = "
  wsF06 <- wsF %>%
    dplyr::filter(dates %in% wsF$dates[stringr::str_detect(string = wsF$dates, pattern = "
  wsF07 <- wsF %>%
```

```
    dplyr::filter(dates %in% wsF$dates[stringr::str_detect(string = wsF$dates, pattern = "
  wsF08 <- wsF %>%
    dplyr::filter(dates %in% wsF$dates[stringr::str_detect(string = wsF$dates, pattern = "
  wsF09 <- wsF %>%
    dplyr::filter(dates %in% wsF$dates[stringr::str_detect(string = wsF$dates, pattern = "
```

7.2 Plotting and testing

```
#
  source("zscripts/z_inputFls.R")
```

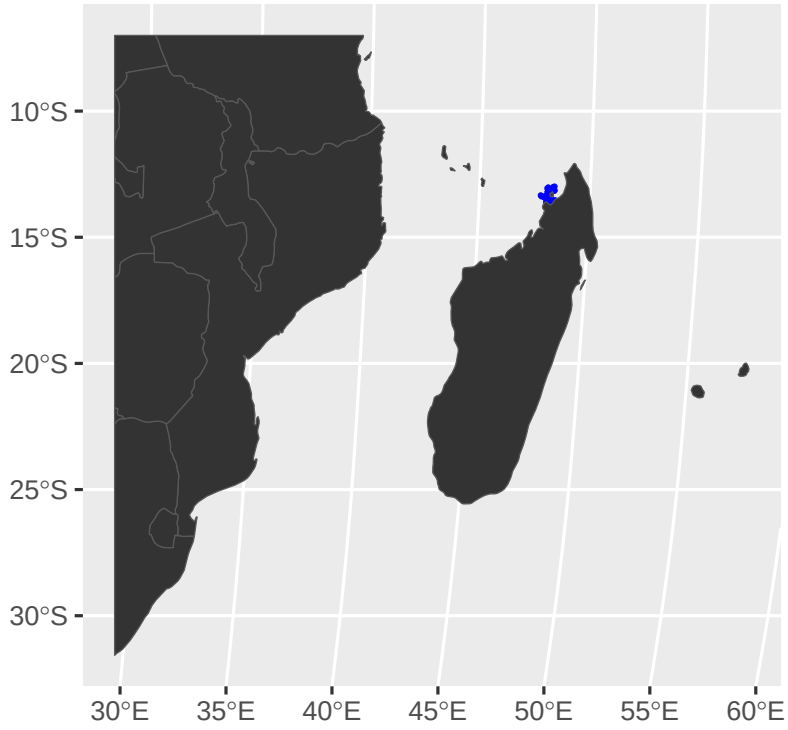
```
terra 1.7.39

Linking to GEOS 3.11.0, GDAL 3.5.3, PROJ 9.1.0; sf_use_s2() is TRUE


Attaching package: 'dplyr'

The following objects are masked from 'package:terra':

    intersect, union

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union


Attaching package: 'tidyr'

The following object is masked from 'package:terra':

    extract
```

```
    source("zscripts/z_helpFX.R") # and the Help function just in case :-)
```

```
Attaching package: 'patchwork'

The following object is masked from 'package:terra':

    area

The legacy packages maptools, rgdal, and rgeos, underpinning the sp package,
which was just loaded, will retire in October 2023.
Please refer to R-spatial evolution reports for details, especially
https://r-spatial.org/r/2023/05/15/evolution4.html.
It may be desirable to make the sf package available;
package maintainers should consider adding sf to Suggests:.
The sp package is now running under evolution status 2
     (status 2 uses the sf package in place of rgdal)

Support for Spatial objects (`sp`) will be deprecated in {rnaturalearth} and will be removed


Attaching package: 'rnaturalearthdata'

The following object is masked from 'package:rnaturalearth':

    countries110


Attaching package: 'gganimate'

The following object is masked from 'package:terra':

    animate


Attaching package: 'transformr'

The following object is masked from 'package:sf':

    st_normalize
```

```
Attaching package: 'data.table'

The following objects are masked from 'package:dplyr':

    between, first, last

The following object is masked from 'package:terra':

    shift

udunits database from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/u

Linking to liblwgeom 3.0.0beta1 r16016, GEOS 3.11.0, PROJ 9.1.0

Loading required package: foreach

Loading required package: gifski

Loading required package: ggnewscale

Loading required package: pals

Warning: attribute variables are assumed to be spatially constant throughout
all geometries
```

```r
# Pick a random object from above
  mmF_test <- wsF01 %>%
    st_transform(crs = robin) # always remember to get a common projection!
# Plot
  p_test <- ggplot() +
    geom_sf(data = mmF_test, colour = "blue", size = 0.3) +
    geom_sf(data = world_sfRob, size = 0.05, fill = "grey20")
  print(p_test)
```



7.3 Near distance to Fronts

```
source("zscripts/z_inputFls.R")
source("zscripts/z_helpFX.R")

# loading libraries
  library(sf)
  library(terra)
  library(stringr)
  library(dplyr)
  library(data.table)
  library(ggplot2)
  library(patchwork)

# defining the arguments
  pus = "data/PUs_MZ_100km2.shp"
  fsle_sf = "data/fsle_pus_100km2/FSLE_SWIO_2011-12.rds"
  fdata = wsF09
```

```
  cutoff = 0.75
  output = "data/"

#
  PUs <- st_read(pus)
  sf1 <- readRDS(fsle_sf)
  nms <- names(sf1) %>%
    stringr::str_extract(pattern = ".*(?=\\.)")
  colnames(sf1) <- nms

# Matching the appropriate Front data set with the megafauna component
# (from fronts data [sf1] pick the same DATES of megafauna data [fdata])
  df01 <- sf1 %>%
    dplyr::select(as.character(unique(fdata$dates)))

# A loop to match data with front and extract which is the closest value
  Fdates <- unique(fdata$dates)
  FF <- vector("list", length = length(Fdates))
  for(i in seq_along(Fdates)) {
  # Filter the megafauna data for each date
    mmF <- fdata %>%
      dplyr::filter(dates == Fdates[i]) %>%
      st_transform(crs = robin)
  # Filter Front data for each date of the megafauna data
    OFCdates <- df01 %>%
      dplyr::select(as.character(Fdates[i]))
    OFCdates <- cbind(PUs, OFCdates) %>%
      st_transform(crs = robin)
  # Estimate the distance to all
    dist02 <- st_distance(mmF, OFCdates, by_element = FALSE) %>%
      t() %>%
      as_tibble()
    colnames(dist02) <- as.character(mmF$ptt)
  # Get the upper front quantile
    qfront <- OFCdates %>%
      as_tibble() %>%
      dplyr::select(2) %>%
      # as.vector() %>%
      quantile(probs = cutoff, na.rm = TRUE) %>%
      as.vector()
  # First 5 FIRST closest distances to a nearest Front
```

```
    dist03 <- apply(X = dist02, MARGIN = 2, FUN = function(x) {
      dist <- x
      final <- cbind(PUs[,1], OFCdates[,2], dist) %>%
        as_tibble() %>%
        dplyr::select(-geometry, -geometry.1) %>%
        dplyr::filter(.[[2]] > qfront) %>%
        dplyr::arrange(.[[3]]) %>%
        dplyr::slice(1:5)
      })
    FF[[i]] <- do.call(cbind, dist03)
  }

# Tidy up the final list
  names(FF) <- Fdates
  FF <- FF[order(names(FF))]

# File name for the output
  ngrd <- unlist(stringr::str_split(basename(pus), "_"))[3] %>%
    stringr::str_remove_all(pattern = ".shp")
  ndate <- unlist(stringr::str_split(basename(fsle_sf), "_"))[3] %>%
    stringr::str_remove_all(pattern = ".rds")
  th <- paste("cutoff", cutoff, sep = "-")
  ffname <- paste("whale-shark_neardist", ngrd, ndate, th, sep = "_")
  saveRDS(FF, paste0(output, ffname, ".rds"))
```

7.4 Clean (yes, even more!)

```
dt <- readRDS("data/whale-shark_neardist_100km2_2011-12_cutoff-0.75.rds")
exm <- lapply(dt, function(x) {
  sngl <- x
  df1 <- split.default(sngl, rep(1:(ncol(sngl)/3), each = 3))
  dist1 <- units::set_units(unlist(lapply(df1, function(x2) x2[1,3])), "m")
  dist2 <- as.vector(dist1)})
Fdates <- paste0(unlist(stringr::str_split(names(exm)[1], pattern = "-"))[1:2], collapse
#
dst1 <- Reduce(c, exm) %>%
  units::set_units("m") %>%
  units::set_units("km")
lsout <- dst1 %>%
```

```
  as_tibble() %>%
  dplyr::mutate(date = as.factor(Fdates)) %>%
  dplyr::rename(neardist = value) %>%
  dplyr::select(date, neardist)
```

7.5 Distance Histograms

```
#
  s1 <- lsout
  Fdates <- as.vector(unique(s1$date))

#
  ff <- ggplot(data = s1,  aes(x = neardist, fill = date)) +
      geom_histogram(data = subset(s1, date == Fdates),
                     aes(x = neardist, y = (..count..)/sum(..count..)),
                     colour = "1",
                     bins = 30) +
      scale_fill_manual(values = c("#2b8cbe"),
                        name = "",
                        labels = Fdates) +
      scale_y_continuous(breaks = seq(0, 1, 0.2), limits = c(0, 1)) +
      coord_cartesian(xlim = c(0, 30)) +
      theme_bw() +
      theme(plot.title = element_text(face = "plain", size = 20, hjust = 0.5),
            plot.tag = element_text(colour = "black", face = "bold", size = 23),
            axis.title.y = element_blank(),
            axis.title.x = element_text(size = rel(1.5), angle = 0),
            axis.text.x = element_text(size = rel(2), angle = 0),
            axis.text.y = element_text(size = rel(2), angle = 0),
            legend.title = element_text(colour = "black", face = "bold", size = 15),
            legend.text = element_text(colour = "black", face = "bold", size = 13),
            legend.key.height = unit(1.5, "cm"),
            legend.key.width = unit(1.5, "cm"),
            legend.position = "none") +
      labs(x = "Distance to high FSLE",
           y = "Density") +
      # geom_richtext(inherit.aes = FALSE,
      #               data = tibble(x = 27, y = 0.8, label = paste("n", "=", nrow(s1), sep
      #               aes(x = x, y = y, label = "label"),
```

```
#               size = 6,
#               fill = NA) +
ggtitle(Fdates)
```

8 Appendix 1: Random Walk analysis in R

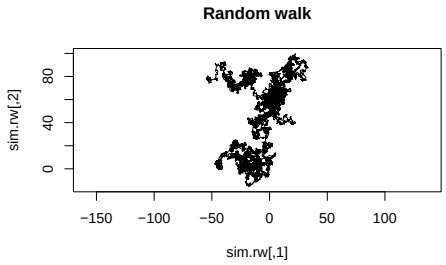
A Random Walk analysis, is a statistical modeling technique used to simulate and analyze sequences of random data points over time.

In a Random Walk, the next data point in the sequence is determined by the current data point plus a random value drawn from a random distribution. The random distribution can be Gaussian (normal), uniform, or any other suitable distribution depending on the application.

8.1 Data import

Source the required dataset.

```
library(SiMRiv)
rand.walker <- species(state.RW())
sim.rw <- simulate(rand.walker, 10000)
plot(sim.rw, type = "l", asp = 1, main = "Random walk")
```



References