Phytoplankton_greater_Baltic_Sea

Daniela Figueroa, Markus Lindh, Luuk van der Heijden, Willem Stolte and Lisa Sundqvist

22-02-2021

Introduction

The large databases of EMODNET Biology only store confirmed presences of species. However, when mapping species distribution, it is also important where the species did not occur: there is at least as much information in absences as in presences. Inferring absences from presence-only databases is difficult and always involves some guesswork. In this product we have used as much meta-information as possible to guide us in inferring absences. There is important meta-information at two different levels: the level of the data set, and the level of the species. Datasets can contain implicit information on absences when they have uniformly searched for the same species over a number of sample locations. Normally, if the species would have been present there, it would have been recorded. Other datasets, however, are not informative at all about absences. Typical examples are museum collections. The fact that a specimen is found at a particular place confirms that it lived there, but does not give information on any other species being present or absent in the same spot. It is also helpful to plot the total number of species versus the total number of samples. Incomplete datasets have far less species than expected for their size, compared to 'complete' datasets. At the species level, taxonomic databases such as WoRMS give information on the functional group the species belongs to. This information is present for many species, but it is incomplete. However, we can use it in a step to select the most useful datasets. We could use it to restrict the species to be used in mapping, but since the data are incomplete that would cause some loss of interesting information.

```
options(tinytex.verbose = TRUE)
knitr::opts_chunk$set(echo = TRUE)
require(raster)
require(sp)
require(rgdal)
require(RColorBrewer)
require(svMisc)
require(tidyverse)
require(sf)
require(worrms)
library(ggplot2)
library(rnaturalearth)
library(magick)
library(rgeos)
library(EMODnetBiologyMaps)
require(rgdal)
require(imis)
require(progress)
##############
downloadDir <- "data/raw_data"
dataDir     <- "data/derived_data"
mapsDir     <- "product/maps"
rasterDir   <- "product/species_rasters"
plotsDir    <- "product/species_plots"
```

```
rareDir     <- "product/rarespecies"
scriptsDir  <- "scripts"
#############
proWG<-CRS("+proj=longlat +datum=WGS84")
#############################################################
# define a raster covering the grid. Set resolution of the raster here
#############################################################
r<-raster(ext=extent(9,37,52,68),ncol=150,nrow=280,crs=proWG,vals=0)
```

Data selection and retrieval

The retrieval of data goes in three steps.

Step 1. Use functional group information to harvest potentially interesting datasets.

A query is performed for data on species known to be phytoplankton (in WoRMS) and to occur in a number of different sea regions. This yields a large dataset with phytoplankton data, but many of these data come from datasets that are not useful for our purpose.

From the datasets resulting from this action we harvest all potentially interesting datasets, that contain at least one phytoplankton species in the region of interest. We subsequently use the imis database with meta-information on the datasets, to list the meta-data of all these datasets. This results in the file ./data/derived_data/allDatasets.csv.

In this file we perform the (manual) selection of datasets to be used. We also qualify the datasets as either 'complete' or 'incomplete'. The result of this manual procedure is the file ./data/derived_data/allDatasets_selection.csv.

```
# read geographic layers for plotting
layerurl <- paste0("http://geo.vliz.be/geoserver/MarineRegions/ows?service=WFS&version=1.0.0&",
                   "request=GetFeature&typeName=MarineRegions:eez_iho_union_v2&",
                   "outputFormat=application/json")
regions <- sf::st_read(layerurl)
# read selected geographic layers for downloading
roi <- read_delim(file.path(dataDir,"regions.csv"), delim = ",")
# check by plotting
regions %>% filter(mrgid %in% roi$mrgid) %>%
  ggplot() +
  geom_sf(fill = "blue", color = "white") +
  geom_sf_text(aes(label = mrgid), size = 2, color = "white") +
  theme(axis.title = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank())
ggsave(file.path(dataDir,"regionsOfInterest.png"), width = 3, height =  4, )
# download data by geographic location and trait
beginDate<- "1900-01-01"
endDate <- "2021-01-01"
attributeID1 <- "phytoplankton"
# Full occurrence (selected columns)
for(ii in 1:length(roi$mrgid)){
  mrgid <- roi$mrgid[ii]
  print(paste("downloading data for", roi$marregion[ii]))
  downloadURL <- paste0(
  "http://geo.vliz.be/geoserver/wfs/ows?service=WFS&version=1.1.0&",
  "request=GetFeature&",
```

```
  "typeName=Dataportal%3Aeurobis-obisenv_full&",
  "resultType=results&",
  "viewParams=where%3A%28%28up.geoobjectsids+%26%26+ARRAY%5B",mrgid,
  "%5D%29%29+AND+%28%28observationdate+BETWEEN+%27",beginDate,
  "%27+AND+%27",endDate,
  "%27+%29%29+AND+aphiaid+IN+%28+SELECT+aphiaid+FROM+eurobis.taxa_attributes+WHERE+selectid+IN+%28%27",
  attributeID1,
  "%27%29%29%3Bcontext%3A0100&",
  "propertyName=datasetid%2C",
  "datecollected%2C",
  "decimallatitude%2C",
  "decimallongitude%2C",
  "scientificname%2C",
  "aphiaid%2C",
  "scientificnameaccepted%2C",
  "yearcollected%2C",
  "waterbody%2C",
  "country%2C",
  "recordnumber%2C",
  "fieldnumber%2C",
  "minimumdepthinmeters%2C",
  "maximumdepthinmeters%2C",
  "aphiaidaccepted%2C",
  "catalognumber%2C",
  "qc%2C",
  "eventid&",
  "outputFormat=csv")
  filename = paste0("region", roi$mrgid[ii], ".csv")
  data <- read_csv(downloadURL,col_types = "cnccTnccccnnnnccccccn")
  write_delim(data, file.path(downloadDir, "byTrait", filename), delim = ",")
}
filelist <- list.files(file.path(downloadDir,"byTrait"))
allDataExtra <- lapply(filelist, function(x)
  read_delim(file.path(downloadDir, "byTrait", x),
             delim = ",",
             col_types = "cnccTnccccnnnnccccccn")) %>%
  set_names(sub(".csv", "", filelist)) %>%
  bind_rows(.id = "mrgid") %>%
  mutate(mrgid = sub("region", "", mrgid))
write_delim(allDataExtra, file.path(dataDir, "allDataExtra.csv"), delim = ",")
# from downloaded data
allDataExtra <- read_delim(file.path(dataDir, "allDataExtra.csv"), delim = ",")
datasetidsoi <- allDataExtra %>% distinct(datasetid) %>%
  mutate(datasetid = as.numeric(sub('http://www.emodnet-biology.eu/data-catalog?module=dataset&dasid=',
                        "", datasetid, fixed = T)))
# retrieve data by dataset
# function to read dataset characteristics
fdr2<-function(dasid){
  datasetrecords <- datasets(dasid)
  dascitations <- getdascitations(datasetrecords)
  if(nrow(dascitations)==0)dascitations<-tibble(dasid=as.character(dasid),title="",citation="")
  if(nrow(dascitations)==1) if(is.na(dascitations$citation)) dascitations$citation<-""
  daskeywords <- getdaskeywords(datasetrecords)
```

```
  if(nrow(daskeywords)==0)daskeywords<-tibble(dasid=as.character(dasid),title="",keyword="")
  if(nrow(daskeywords)==1) if(is.na(daskeywords$keyword))daskeywords$keyword<-""
  dascontacts <- getdascontacts(datasetrecords)
  if(nrow(dascontacts)==0)dascontacts<-tibble(dasid=as.character(dasid),title="",contact="")
  if(nrow(dascontacts)==1) if(is.na(dascontacts$contact))dascontacts$contact<-""
  dastheme <- getdasthemes(datasetrecords)
  if(nrow(dastheme)==0)dastheme<-tibble(dasid=as.character(dasid),title="",theme="")
  if(nrow(dastheme)==1) if(is.na(dastheme$theme))dastheme$theme<-""
  dastheme2 <- aggregate(theme ~ dasid, data = dastheme, paste,
                         collapse = " , ")
  daskeywords2 <- aggregate(keyword ~ dasid, data = daskeywords,
                            paste, collapse = " , ")
  dascontacts2 <- aggregate(contact ~ dasid, data = dascontacts,
                            paste, collapse = " , ")
  output <- dascitations %>% left_join(dascontacts2, by = "dasid") %>%
    left_join(dastheme2, by = "dasid") %>% left_join(daskeywords2,
                                                     by = "dasid")
  return(output)
}
all_info <- data.frame()
for (i in datasetidsoi$datasetid){
  dataset_info <- fdr2(i)
  all_info <- rbind(all_info, dataset_info)
}
names(all_info)[1]<-"datasetid"
write_delim(all_info, file=file.path(dataDir,"allDatasets.csv"), delim = ",")
# Note: this step is followed by manual inspection of data sets, and selection
# results in file "./data/derived_data/allDatasets_selection.csv"
```

Step 2. Download by dataset.

In this step, we download the part of all these useful datasets that occur in the region of interest. For practical reasons this region is subdivided in many subregions - in that way the downloaded files are not too big and there is less risk of interruptions of the process. After download, all these files will be recombined into one big datafile.

```
# read selected geographic layers for downloading
roi <- read_delim(file.path(dataDir,"regions.csv"), delim = ",")
getDatasets <- read_delim(file.path(dataDir,"allDatasets_selection.csv"), delim = ",")
getDatasets <- getDatasets %>%
  filter(include)
for(ii in 1:length(roi$mrgid)){
  for(jj in 1:length(getDatasets$datasetid)){
    datasetid <- getDatasets$datasetid[jj]
    mrgid <- roi$mrgid[ii]
    print(paste("downloading data for ", roi$marregion[ii], "and dataset nr: ", datasetid))
    downloadURL <- paste0("https://geo.vliz.be/geoserver/wfs/ows?service=WFS&version=1.1.0",
      "&request=GetFeature&typeName=Dataportal%3Aeurobis-obisenv_full&resultType=results&",
      "viewParams=where%3A%28%28up.geoobjectsids+%26%26+ARRAY%5B", mrgid,
      "%5D%29%29+AND+datasetid+IN+(",datasetid,");context%3A0100&propertyName=datasetid%2C",
      "datecollected%2Cdecimallatitude%2Cdecimallongitude%2Ccoordinateuncertaintyinmeters%2C",
      "scientificname%2Caphiaid%2Cscientificnameaccepted%2Cinstitutioncode%2Ccollectioncode%2C",
      "occurrenceid%2Cscientificnameauthorship%2Cscientificnameid%2Ckingdom%2Cphylum%2Cclass",
      "%2Corder%2Cfamily%2Cgenus%2Csubgenus%2Caphiaidaccepted%2Cbasisofrecord%2Ceventid&",
```

```
      "outputFormat=csv")
    data <- read_csv(downloadURL, col_types = "ccccccTnnnccccccccccccccc")
    filename = paste0("region", roi$mrgid[ii], "_datasetid", datasetid,  ".csv")
    if(nrow(data) != 0){
      write_delim(data, file.path(downloadDir, "byDataset", filename), delim = ",")
    }
  }
}
```

step 3. Combine all downloaded datasets into one big dataset

In this step, we read in all the files written during the previous step, and combine the data into one big dataset to be used for further analysis and production of the maps. It is kept separate from the previous step because the downloading is very time-consuming and it can better be completed before this step takes place, including checks on errors, timeouts etc.

```
filelist <- list.files(file.path(downloadDir,"byDataset"))
all2Data <- lapply(filelist, function(x)
  read_delim(file.path(downloadDir,"byDataset", x),
             delim = ",",
             col_types = "ccccccTnnnccccccccccccccc"
            )
) %>%
  set_names(filelist) %>%
  bind_rows(.id = "fileID") %>%
  separate(fileID, c("mrgid", "datasetID"), "_") %>%
  mutate(mrgid = sub("[[:alpha:]]+", "", mrgid)) %>%
  mutate(datasetID = sub("[[:alpha:]]+", "", datasetID))
# mutate(mrgid = sub("region", "", mrgid))
all2Data<- all2Data %>%
  mutate(AphiaID=as.numeric(substr(aphiaidaccepted,52,65)))%>%
  filter(!is.na(AphiaID)) %>%
  filter(!is.na(decimallongitude)) %>%
  filter(!is.na(decimallatitude)) %>%
  filter(!is.na(datecollected))
write_delim(all2Data, file.path(dataDir, "all2Data.csv"), delim = ",")
```

Analysis of the species represented in the dataset

The selection of data in the first step makes use of the traits stored in WoRMS, the World Register of Marine Species. For many species in this database, the functional group to which the species belongs is recorded. However, this is not yet complete. We can help the development of these traits databases from the compilation of data performed here. Since we selected phytoplankton data sets, we can assume that most species in our database will be phytoplankton, although it appears this is not absolutely the case everywhere. Here we try to use as much information as possible, either from the traits database or from the taxonomic position of the taxa, to derive what functional group they belong to. That is used to narrow down the list of taxa to the phytoplankton species we are targeting, but also to report back to WoRMS with suggestions to improve the traits database.

Results have been written to a file after performing this step once.

```
# we build the species list, keeping the taxonomic information we have in the total data set
# we foresee logical columns in the species list to group the species by in the rest of this script
bns<-read_delim(file.path(dataDir,"all2Data.csv"),
                col_types = "cccccccTnnnccccccccccccccccn",
```

```
                delim=",")
splst <- bns %>%
         select(AphiaID,scientificnameaccepted,phylum,class,order,family,genus,subgenus) %>%
         distinct() %>%
         mutate(benthos=FALSE,endobenthos=FALSE,macrobenthos=FALSE,epibenthos=FALSE, meiobenthos=FALSE,
# determine, using attributes, which species are phytoplankton
# (done once, result stored as delimited file)
# nsp_attr<-data.frame(AphiaID=NA,measurementTypeID=NA,measurementType=NA,
#                      measurementValue=NA,source_id=NA,reference=NA,qualitystatus=NA,
#                      AphiaID_Inherited=NA,CategoryID=NA)
nsp_attr<-tibble()
for(i in 1:nrow(splst)){
  print(paste(i,"out of",nrow(splst),"downloading attributes of species",
              splst$scientificnameaccepted[i],"AphiaID",splst$AphiaID[i]))
  ttt<-NULL
  try(ttt<-wm_attr_data(id=splst$AphiaID[i],include_inherited = T),silent = T)
  if(! is.null(ttt)) nsp_attr<-rbind(nsp_attr,ttt[,1:9])
}
nsp_attr <- nsp_attr %>%
                  mutate(AphiaID=as.numeric(AphiaID)) %>%
                  left_join(splst,by="AphiaID")
write_delim(nsp_attr,file.path(dataDir,"nsp_attr.Rdata"),delim=",")
#
#nsp_attr <- read_delim(file.path(dataDir,"nsp_attr.csv"),delim=",")
#
# what Functional groups are there?
fg <- nsp_attr %>% filter(measurementType=="Functional group") %>%
                   select(measurementValue) %>%
                   distinct
print(fg)
# what Paraphyletic groups are there?
pfg <- nsp_attr %>% filter(measurementType=="Paraphyletic group") %>%
                    select(measurementValue) %>%
                    distinct
print(pfg)
# fill in attributes columns of splst based on the attributes downloaded from WoRMS
set_attr<-function(attr){
  tt <- nsp_attr                                  %>%
          filter(grepl(attr,measurementValue)) %>%
          select(AphiaID)                       %>%
          distinct()
  splst <- splst %>%
             mutate(!!attr:=ifelse(AphiaID %in% tt$AphiaID,TRUE,FALSE))
  return(splst)
}
splst<-set_attr("Algae")
# fill in attributes columns based on taxonomic information
splst$Algae_tax  <- splst$Algae_tax  | splst$phylum == "Chlorophyta"
splst$Algae_tax  <- splst$Algae_tax  | splst$phylum == "Rhodophyta"
splst$Algae_tax  <- splst$Algae_tax  | splst$phylum == "Ochrophyta"
splst$Algae_tax  <- splst$Algae_tax  | splst$phylum == "Charophyta"
splst$Algae_tax  <- splst$Algae_tax  | splst$phylum == "Cyanobacteria"
splst$Algae_tax  <- splst$Algae_tax  | splst$phylum == "Haptophyta"
```

```
# write splst to output
sp2use<- splst %>%
  filter(Algae)
write_delim(splst,file.path(dataDir,"sp2use.csv"),delim=",")
write_delim(sp2use,file.path(dataDir,"sp2use.csv"),delim=",")
```

Production of rasters with presence/absence information

For the production of maps, we define 'sampling events' as the ensembles of records that share time and place. We consider such events as one sample. For the incomplete datasets, we inventory what species they have targeted. Finally, for every species we determine whether or not it was present in all sampling events of all relevant datasets. This presence/absence information is written to the output file, together with the spatial location and the sampling date. The intention is to use this information to produce interpolated maps covering also the non-sampled space. As a first step in visualisation, we rasterize this information and show it in a map per species. In the following code block, the rasters are calculated and stored in one file per species (for all species that were found more than 200 times - this could easily be extended to all species).

```
############################################################
#### load data
############################################################
sbnsc<- read_delim(file.path(dataDir,"all2Data.csv"),
                   col_types = "ccccccccTnnnccccccccccccccccn",
                   delim=",")
trdi<-read.csv("./data/derived_data/allDatasets_selection.csv",
               stringsAsFactors = FALSE)
usedds<- trdi %>% filter (include) %>% dplyr::select(datasetid)
splst<-read_delim(file.path(dataDir,"sp2use.csv"),
                  col_types = "dccccccclllllllllllllllll",
                  delim=",")
############################################################
##### select few columns to work with
##### filter to only the used datasets
##### and filter to true phytoplankton species only
############################################################
trec<- sbnsc %>% dplyr::select(eventDate=datecollected,
                               decimalLongitude=decimallongitude,
                               decimalLatitude=decimallatitude,
                               scientificName=scientificnameaccepted,
                               aphiaID=AphiaID,
                               datasetid=datasetid) %>%
  mutate(datasetid=as.numeric(substr(datasetid,65,90))) %>%
  filter(datasetid %in% usedds$datasetid)
trec<- trec %>%  filter(aphiaID %in% splst$AphiaID)
############################################################
# Define 'sampling events' as all records that share time and place, give
# ID numbers to all events (eventNummer), and store the eventNummer in each
# record of trec
############################################################
events<- trec %>% dplyr::select(eventDate,decimalLongitude,decimalLatitude) %>%
  distinct() %>%
  mutate(eventNummer=row_number())
trec <- trec %>% left_join(events,by=c('eventDate','decimalLongitude','decimalLatitude'))
########### work on datasets
#
```

```
#### check on completeness
#
nsp<-trec %>% group_by(datasetid) %>%
  distinct(aphiaID)    %>%
  mutate(nspec=n())    %>%
  dplyr::select(datasetid,nspec) %>%
  distinct()
nev<-trec %>% group_by(datasetid)     %>%
  distinct(eventNummer)    %>%
  mutate(nev=n())          %>%
  dplyr::select(datasetid,nev) %>%
  distinct()                    %>%
  left_join(nsp,by='datasetid')%>%
  left_join(trdi,by='datasetid')
#
plot(nev$nev,nev$nspec,log="xy",col=ifelse(nev$complete,"blue","red"),pch=19,
     xlab="number of events in dataset",ylab="number of species in dataset")
text(nev$nev*1.2,nev$nspec*(1+(runif(nrow(nev))-0.5)*0.4),nev$datasetid,cex=0.5)
#
# manage the incomplete datasets
#
trdi_ct<-trdi %>% filter (complete)
trdi_ic<-trdi %>% filter (!complete)
# make a species list for each incomplete dataset
ic_sp<-data.frame(datasetid=NULL,aphiaID=NULL)
for(i in 1:nrow(trdi_ic)){
  ds<-trdi_ic$datasetid[i]
  specs<-unique(trec$aphiaID[trec$datasetid==ds])
  ic_sp<-rbind(ic_sp,data.frame(datasetid=rep(ds,length(specs)),aphiaID=specs))
}
#################################################################
# find occurrence frequency of all species, and rank the species accordingly
#
spfr<- trec %>%
  group_by(aphiaID,scientificName) %>%
  summarize(n_events=n()) %>%
  arrange(desc(n_events))
nsptoplot<-length(which(spfr$n_events>200))
############ end of the generic part. What follows is a loop over the species ##
spmin<-1
spmax<-nsptoplot
pb <- progress_bar$new(total=nsptoplot)
for(ss in spmin:spmax){
  spAphId<-spfr$aphiaID[ss]
  specname<-spfr$scientificName[ss]
  spcolumn<-paste0("pa",spAphId)
  pb$tick()
  # # from the list of incomplete datasets, check if they have our species.
  # # Only keep these, drop the others
  # tt_ds<- ic_sp                       %>%
  #   filter(aphiaID==spAphId) %>%
  #   distinct(datasetid)      %>%
  #   bind_rows(trdi_ct        %>%
```

```
  #               dplyr::select(datasetid))
  # The dataset to be used consists of all complete datasets, and all
  # incomplete datasets that targeted our species
  spe<- trec                                                          %>%
#    filter(datasetid %in% tt_ds$datasetid)                  %>%
    group_by(eventNummer)                                    %>%
    summarize(pres_abs= as.numeric(any(aphiaID==spAphId)),.groups = 'drop')  %>%
    left_join(events,by='eventNummer')
  spesh <- spe                                  %>%
    mutate (spcolumn=(pres_abs==1)) %>%
    dplyr::select (- pres_abs)
  names(spesh)[5]<-spcolumn
  spesh <- spesh %>% dplyr::select('eventNummer',spcolumn)
  if(ss==spmin) allspe <- spesh else {
    allspe <- allspe %>% full_join(spesh,by='eventNummer')
  }
  coordinates(spe)<- ~decimalLongitude+decimalLatitude
  projection(spe)<-proWG
  r1<-rasterize(spe,r,field="pres_abs",fun=mean)
  # Export rasters as tif
  raster::writeRaster(
    r1,
    file.path(
      rasterDir, paste0(
        sprintf("%04d",ss), "_",
        spAphId, "_",
        gsub(" ", "-", specname),
        ".tif"
      )
    ),
    overwrite=TRUE
  )
}
# save file with presence/absence information of all species plotted
evs<-tibble(eventNummer=allspe$eventNummer)
evs<- evs %>% left_join(events,by='eventNummer')
spe<- cbind(evs,allspe[,2:ncol(allspe)])
save(spe,file=file.path(mapsDir,"spe.Rdata"))
write_delim(spfr,path=file.path(mapsDir,"specieslist.csv"),delim=",")
for(i in 5:ncol(spe)) spe[,i]<-as.numeric(spe[,i])
write_delim(spe,path=file.path(mapsDir,"spe.csv"),delim=",")
```

Production of maps of the rasters

We use the EMODnet mapping package EMODnetBiologyMaps to produce maps of the rasters.

```
# correct the EMODnetBiologyMaps package by running a slightly modified version
emodnet_map_plot_2<-function (data, fill = NULL, title = NULL, subtitle = NULL,
                              legend = NULL, crs = 3035, xlim = c(2426378.0132,
                              7093974.6215), ylim = c(1308101.2618, 5446513.5222),
                              direction = 1, option = "viridis",zoom = FALSE, ...)
{
  stopifnot(class(data)[1] %in% c("sf", "RasterLayer", "SpatialPolygonsDataFrame",
                                  "SpatialPointsDataFrame","SpatialLinesDataFrame"))
```

```
  emodnet_map_basic <- emodnet_map_basic(...) + ggplot2::ggtitle(label = title,
                                                                 subtitle = subtitle)
  if (class(data)[1] == "RasterLayer") {
    message("Transforming RasterLayer to sf vector data")
    data <- sf::st_as_sf(raster::rasterToPolygons(data))
    fill <- sf::st_drop_geometry(data)[, 1]
  }
  if (class(data)[1] %in% c("SpatialPolygonsDataFrame",
                            "SpatialPointsDataFrame", "SpatialLinesDataFrame")) {
    message("Transforming sp to sf")
    data <- sf::st_as_sf(data)
  }
  if (sf::st_geometry_type(data, FALSE) == "POINT") {
    emodnet_map_plot <- emodnet_map_basic + ggplot2::geom_sf(data = data,
                          color = emodnet_colors()$yellow)
  }
  if (sf::st_geometry_type(data, FALSE) == "POLYGON") {
    emodnet_map_plot <- emodnet_map_basic +
      ggplot2::geom_sf(data = data,ggplot2::aes(fill = fill),
                       size = 0.05,color=NA) +
      ggplot2::scale_fill_viridis_c(alpha = 0.8,
  }
  if (zoom == TRUE) {
    bbox <- sf::st_bbox(sf::st_transform(data, crs))
    xlim <- c(bbox$xmin, bbox$xmax)
    ylim <- c(bbox$ymin, bbox$ymax)
  }
  emodnet_map_plot <- emodnet_map_plot + ggplot2::coord_sf(crs = crs,
                                                           xlim = xlim, ylim = ylim)
  return(emodnet_map_plot)
}
# end of modified plot function - eventually to be moved into the package
# load specieslist
spfr<-read_delim(file.path(mapsDir,"specieslist.csv"),delim=",")
nsptoplot<-length(which(spfr$n_events>200))
spmin<-1
spmax<-nsptoplot
pb <- progress_bar$new(total=nsptoplot)
for(ss in spmin:spmax){
  pb$tick()
  spAphId<-spfr$aphiaID[ss]
  specname<-spfr$scientificName[ss]
  rasterfil <- file.path(rasterDir,
     paste0(sprintf("%04d",ss), "_",spAphId, "_",gsub(" ", "-", specname),".tif"))
  r1<-raster(rasterfil)
  legend="P(pres)"
  # Plot the grid

  ec<-emodnet_colors()
  plot_grid <- emodnet_map_plot_2(data=r1,title=specname,subtitle=paste0('AphiaID ', spAphId),
                                  zoom=TRUE,seaColor=ec$darkgrey,landColor=ec$lightgrey,legend=legend)
  filnam<-file.path(plotsDir,
                    paste0(sprintf("%04d",ss), "_",spAphId, "_",gsub(" ", "-", specname),".png"))
```

```
  emodnet_map_logo(plot_grid,path=filnam,width=120,height=160,dpi=300,units="mm",offset="+0+0")

}
```

Reproducibility

```
# Date time
Sys.time()
```

```
## [1] "2021-02-22 09:20:58 CET"
```

```
# Here we store the session info for this script
sessioninfo::session_info()
```

```
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.0.3 (2020-10-10)
##  os       Windows 10 x64
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  Swedish_Sweden.1252
##  ctype    Swedish_Sweden.1252
##  tz       Europe/Berlin
##  date     2021-02-22
##
## - Packages -------------------------------------------------------------------
##  package     * version date       lib source
##  assertthat    0.2.1   2019-03-21 [1] CRAN (R 4.0.3)
##  cli           2.2.0   2020-11-20 [1] CRAN (R 4.0.3)
##  crayon        1.3.4   2017-09-16 [1] CRAN (R 4.0.3)
##  digest        0.6.27  2020-10-24 [1] CRAN (R 4.0.3)
##  evaluate      0.14    2019-05-28 [1] CRAN (R 4.0.3)
##  fansi         0.4.2   2021-01-15 [1] CRAN (R 4.0.3)
##  glue          1.4.2   2020-08-27 [1] CRAN (R 4.0.3)
##  htmltools     0.5.1.1 2021-01-22 [1] CRAN (R 4.0.3)
##  knitr         1.31    2021-01-27 [1] CRAN (R 4.0.3)
##  magrittr      2.0.1   2020-11-17 [1] CRAN (R 4.0.3)
##  rlang         0.4.10  2020-12-30 [1] CRAN (R 4.0.3)
##  rmarkdown     2.6     2020-12-14 [1] CRAN (R 4.0.3)
##  sessioninfo   1.1.1   2018-11-05 [1] CRAN (R 4.0.3)
##  stringi       1.5.3   2020-09-09 [1] CRAN (R 4.0.3)
##  stringr       1.4.0   2019-02-10 [1] CRAN (R 4.0.3)
##  withr         2.4.1   2021-01-26 [1] CRAN (R 4.0.3)
##  xfun          0.20    2021-01-06 [1] CRAN (R 4.0.3)
##  yaml          2.2.1   2020-02-01 [1] CRAN (R 4.0.3)
##
## [1] C:/__R/R-4.0.3/library
```