Step 1.1 Creating field-based plots

1.1.1. Creating plots based on original samples

Diane ESPEL

2025-07-22

Contents

1 Objectives

This R script aims to generate square quadrat plots centered on ecological sampling points collected in the field from both recent and historical sources. The workflow includes:

- Loading reference field sample files for different sources (recent `"FIELD"` and historical `"HFI"`).
- Projecting geographic coordinates to a local metric system (EPSG:32739)
- Computing square buffer areas around the points
- Exporting both sampling centroids and polygon quadrats as shapefiles for further spatial analysis

2 Script explanation

2.1 Clean environment and graphics

```
rm(list = ls()) # Clear all objects from the R environment to start fresh
graphics.off()   # Close all graphics devices (if any plots are open)
```

2.2 Load required packages

```r
library(sp)          # For SpatialPolygons and Spatial data structures.
library(sf)          # For modern handling of spatial vector data.
library(dplyr)       # For data manipulation (filtering, joining, etc.).
library(data.table)  # For fast file reading and merging.
```

2.3 Define Global Variables

Note: A global variable is a variable defined outside of any function. This means the variable is accessible from any part of the code, including inside functions. A global variable retains its value throughout the execution of the R script unless it is explicitly modified in the code.

It is important to define at a minimum:

- the `"District"`: the archipelago of interest (e.g. "CRO" for Crozet archipelago)
- the `"Island"`: the island within the archipelago of interest (e.g. "POS" for Possession island)

```r
District='CRO' # 3-letter code for the archipelago
Island='POS'   # 3-letter code for the island
```

2.4 Set working directory

You must define a general root directory (`"localHOME"`) that serves as the base path for your input and output data. This directory should point to the local environment where:

- input sample files are located under `"/data/samples"`
- outputs (e.g., plots or shapefiles) will be saved under `"/data/vector/Plots/PrimaryTypo"`

You can create additional subfolders within this structure to organize your outputs by data type, source, or year of analysis.

```r
# Define local root directory
#localHOME = paste0("your_local_path/")
localHOME=paste0("/home/genouest/cnrs_umr6553/despel/CARTOVEGE/")

# Define path to load input sample data
open_samples_path=paste0(localHOME,"data/samples")

# Define path to save generated outputs (plots)
save_plots_path=paste0(localHOME,"data/vector/Plots/PrimaryTypo")
```

2.5 Create field reference samples plots

The script processes two sources: `"FIELD"` (recent field samples) and `"HFI"` (historical field samples) in 7 main steps:

1) Load and merge Sample Files

- List files corresponding to each source.
- Load and concatenate the sample datasets.
- Remove duplicates and assign source information.

2) Apply Coordinate Projection

- Convert longitude and latitude from EPSG:4326 to EPSG:32739 (UTM zone).
- Add new projected coordinates (X and Y in meters).

3) Prepare Final Table

- Select and rename key columns: observation ID, coordinates, surface, habitat levels, etc.

4) Compute Quadrat Corner Coordinates

- Define quadrat size (e.g., 4 meters side).
- Compute coordinates for corners of each square (N, S, E, W).
- Ensure no duplicated observation IDs.

5) Save Sampling Centroids

- Export updated sampling point data (with projected coordinates) to CSV.

6) Draw and Save Quadrats

- Generate square polygons clockwise from NW corner.
- Build `SpatialPolygons` assigning each an ID and then a `SpatialPolygonsDataFrame`.
- Join back sample attributes to each polygon from the original observations (coordinates, date, source, habitat levels, etc.).
- Convert the quadrats to `sf` object and export to shapefile.

7) Export Results

- Export quadrat polygons and centroid points as `.shp` files using `st_write()` from the `sf` package.

```r
# Define list of sources: FIELD = recent samples, HFI = historical samples
source_list=list("FIELD","HFI")

# Loop over source list
for (source in source_list){

  #source="FIELD" # For debugging

  print(paste0("Working with source: ", source))

  # Open and prepare data ------------------------------------------------------

  # Build file name pattern for input CSV files
  pattern <- paste0("Classified_", District, "_", Island, "_", source,
                    "_SAMPLES_",".*.csv")

  # List all matching files in the samples folder
  all_files <- list.files(path = open_samples_path, pattern = pattern,
                          full.names = TRUE)

  print(all_files) # Print the list of matched filenames
```

```r
file_list <- list()
for (i in 1:length(all_files)) {
  file_list[[i]] <- read.csv(all_files[i], sep = ";", dec = ".", stringsAsFactors = FALSE) # Read
↪  each file and store in list
  assign(paste0(source,i), read.csv(all_files[i], sep = ";", dec = ".", stringsAsFactors =
   ↪  FALSE)) # Assign file to dynamic variable
}

Nb_database = length(all_files) # Count number of files imported
print(paste0("Number of imported databases: ", Nb_database))

# Merge all dataframes into a single one (even if only one input file)
df = rbindlist(file_list)

# Remove potential duplicates
New_df = unique(df)
New_df = as.data.frame(New_df)

# Add source identifier
New_df$Source = source

print(paste0("Number of unique observations: ", length(unique(New_df$N_observation))))

# Convert XY coordinates columns --------------------------------------------

filt_points <- New_df

# Convert coordinates from EPSG 4326 (decimal degrees) to EPSG 32739 (UTM Zone 39S)
print("Columns Longitude_ddd and Latitude_ddd are in EPSG 4326 (decimal degrees) and must be
 ↪  converted to EPSG 32739 (meters)")
filt_points <- st_as_sf(filt_points, coords = c("Longitude_ddd", "Latitude_ddd"), crs = 4326) %>%
  st_transform(32739)

# Extract converted coordinates (X and Y in meters)
filt_points_m <- st_coordinates(filt_points)

# Re-add longitude and latitude columns
filt_points$xcoord_m <- filt_points_m[, "X"]
filt_points$ycoord_m <- filt_points_m[, "Y"]
filt_points$Longitude_ddd <- New_df[, "Longitude_ddd"]
filt_points$Latitude_ddd <- New_df[, "Latitude_ddd"]

# Select columns of interest
final_points <- filt_points[, c("N_observation", "Date_manip", "New_protocole","Source",
↪  "Longitude_ddd", "Latitude_ddd", "xcoord_m","ycoord_m", "Aire_m2", "Longueur_m", "Largeur_m",
↪  "Habitat_L1", "Habitat_L2", "Habitat_L3", "Habitat_L4")]

# Rename columns for clarity
colnames(final_points) <- c("N_obs", "Date", "Protocole","Source", "Longitude",
 ↪  "Latitude","xcoord_m","ycoord_m", "Surface", "Longueur_m", "Largeur_m", "Hab_L1", "Hab_L2",
 ↪  "Hab_L3", "Hab_L4","geometry")

# Create spatial polygons  --------------------------------------------------

# Compute corners of quadrat based on center (coordinates) and desired size
print("Defining quadrat side length (m) and calculating corner coordinates")
length_quadrat <- 4
radius <- length_quadrat / 2
poly_points <- data.frame(final_points)
```

```r
poly_points$yPlus <- poly_points$ycoord_m + radius  # North
poly_points$xPlus <- poly_points$xcoord_m + radius  # East
poly_points$yMinus <- poly_points$ycoord_m - radius # South
poly_points$xMinus <- poly_points$xcoord_m - radius # West

print("Check for duplicate observation IDs (should be unique)")
indices_doublons <- which(duplicated(poly_points$N_obs) | duplicated(poly_points$N_obs, fromLast
↪  = TRUE))
print(indices_doublons)



# Construct quadrat polygons (5 points: NW to NW clockwise to close the shape)
quadrats <- cbind(poly_points$xMinus, poly_points$yPlus,   # NW
                  poly_points$xPlus, poly_points$yPlus,    # NE
                  poly_points$xPlus, poly_points$yMinus,   # SE
                  poly_points$xMinus, poly_points$yMinus,  # SW
                  poly_points$xMinus, poly_points$yPlus)   # back to NW

# Create spatial polygons from coordinates
ID <- poly_points$N_obs # Extract observation IDs
print("Creating spatial polygons (quadrat plots)")
polysHabitat <- SpatialPolygons(
  mapply(function(poly, id) {
    xy <- matrix(poly, ncol=2, byrow=TRUE)      # Create coordinate matrix
    Polygons(list(Polygon(xy)), ID = id)        # Create Polygon object with ID
  },
  split(quadrats, row(quadrats)), ID),          # Split coordinates by row (one polygon per row)
  proj4string = CRS("+init=EPSG:32739")         # Define projection
)

plot(polysHabitat)  # Plot polygons for visual check

# Create SpatialPolygonsDataFrame to store attributes
# This object links each quadrat polygon with its corresponding metadata (attributes)
polys.df <- SpatialPolygonsDataFrame(polysHabitat,  # The SpatialPolygons object containing
↪  geometry
                                     data.frame(N_obs = ID, row.names = ID) # Create a minimal
                                      ↪  data.frame with IDs as row names
                                     )
polys.df$xcoord_m = poly_points$xcoord_m # Add additional attribute fields to the
↪  SpatialPolygonsDataFrame
polys.df$ycoord_m = poly_points$ycoord_m
polys.df$Longitude = poly_points$Longitude
polys.df$Latitude = poly_points$Latitude
polys.df$Date = poly_points$Date
polys.df$Protocole = poly_points$Protocole
polys.df$Source = poly_points$Source
polys.df$Surface = poly_points$Surface
polys.df$Longueur_m = poly_points$Longueur_m
polys.df$Largeur_m = poly_points$Largeur_m
polys.df$Hab_L1 = poly_points$Hab_L1
polys.df$Hab_L2 = poly_points$Hab_L2
polys.df$Hab_L3 = poly_points$Hab_L3
polys.df$Hab_L4 = poly_points$Hab_L4

# Save the final shapefiles  -------------------------------------------------

# Export polygons as shapefile
```

```r
  polys.sf <- st_as_sf(polys.df)
  st_write(polys.sf, paste0(save_plots_path, "/Quadrats_", District, "_", Island, "_ALL_", source,
   ↪  "_SAMPLES_Polygons_EPSG32739.shp"), driver = "ESRI Shapefile", append = FALSE)

  # Get centroids from polygons
  pts.in.polys <- sf::st_centroid(polys.sf)
  st_write(pts.in.polys, paste0(save_plots_path, "/Quadrats_", District, "_", Island, "_ALL_",
   ↪  source, "_SAMPLES_Centroids_EPSG32739.shp"), driver = "ESRI Shapefile", append =FALSE)

    # Save centroid coordinates as CSV
  FILE3 = paste0(save_plots_path, "/Quadrats_", District, "_", Island, "_ALL_", source,
 ↪  "_SAMPLES_Centroids_EPSG32739.csv")
  write.table(poly_points, file = FILE3, sep = ";", dec = ".", row.names = FALSE)


} # end of source loop
```

This script enables the spatial representation of ecological sampling points and prepares geospatial layers suitable for GIS analysis or habitat modeling.