Step 1.2 Creating photo-interpreted plots

1.2.2 Getting random points within polygon-based observed map

Diane ESPEL

2025-06-18

Contents

1 Objectives

This script aims to compute 3 types of points within polygons-based map: - **Pole of Inaccessibility (POI)** for each polygon, which corresponds to a random internal point located at a defined distance from the boundary of a polygon of interest. - **Point on surface** for each polygon, representing the centroid for irregular polygons (i.e., those with random shapes). - A defined number of **random points** for large polygons (`"big_polygons_map_sf"`) (i.e., those with an area greater than X m²).

Coordinates are calculated in both EPSG:32739 and EPSG:4326.

2 Script explanation

2.1 Clean environment and graphics

```r
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

```
library(sp)  # Classes and methods for handling spatial data (legacy package)
library(sf)  # Simple Features for spatial data - modern and efficient spatial data package
library(polylabelr)  #
library(dplyr)  # Data manipulation functions like filter, join, select, mutate, etc.
```

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)
- the `"Satellite1"`: the name of the satellite used for photo-interpretation
- the `"Year1"`: the year of imagery acquisition (i.e. the photo-interpretation year)

```
District = "CRO"  # 3-letter code for the archipelago
Island = "POS"  # 3-letter code for the island
Satellite1 = "Pleiades"  # Name of satellite used for photo-interpretation
Year1 = "2022"  # Year of imagery acquisition
```

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 polygon map of observed habitats is located under `"data/vector/Observed_map/PrimaryTypo"`
- outputs of random points will be saved under `"data/vector/Observed_map/PrimaryTypo"`

```
# Local path localHOME = paste0('your_local_path/')
localHOME = paste0("/home/genouest/cnrs_umr6553/despel/CARTOVEGE/")

# path where to open observed map
open_ObsMap_path = paste0(localHOME, "data/vector/Observed_map/PrimaryTypo")

# path where to save your results
save_ObsMap_path = paste0(localHOME, "data/vector/Observed_map/PrimaryTypo")
```

2.5 Load an prepare polygons map

Users can work either on the original observed map (`files_observed`), or on the corrected one (`files_corrected`) (see **optional 1.2.1 substep**).

```r
# List all shapefiles matching expected filename pattern
files_corrected <- list.files(open_map_path, pattern = paste0("^Corrected_observed_map_",
    District, "_", Island, "_", Satellite1, "_", Year1, "_EPSG32739\\.shp$"), full.names = TRUE)
files_observed <- list.files(open_map_path, pattern = paste0("^Observed_map_", District,
    "_", Island, "_", Satellite1, "_", Year1, "_EPSG32739\\.shp$"), full.names = TRUE)

# If a corrected file exists, use it; otherwise, use the uncorrected version
# (if it exists)
if (length(files_corrected) > 0) {
    FILE1 <- files_corrected[1]
} else if (length(files_observed) > 0) {
    FILE1 <- files_observed[1]
}
polygons_map <- st_read(FILE1)

# Convert SpatialPolygons into sf object
polygons_map_sf = st_as_sf(polygons_map)
```

2.6 Get Poles of Inaccessibility

Random internal points are calculated at a specified distance from the boundaries of the polygons. To achieved this :

-The `poi()` function returns the geographic coordinates of random internal points (`poles`) and combines the results into a single data table.

- A minimum `distance` from the polygon boundaries is applied to filter the `poles` and the filtered poles are converted to a spatial object `poles_sf` with specific coordinate names and unique identifiers `id`.
- The script then intersects the poles with the original polygons `polygons_map_sf` to extract relevant attributes. It adds coordinates in both EPSG:32739 and EPSG:4326 formats and prepares the final dataset by including additional metadata such as date and source.

```r
print("Computing coordinates of poles of inaccessibility for all polygons")

# Compute pole of inaccessibility
poles <- poi(polygons_map_sf)
poles <- data.table::rbindlist(poles)

# Convert poles to spatial object and filter them according to distance from polygon boundaries
distance = 5 # Define threshold distance from polygons boundaries
poles_sf <- poles %>%
  filter(dist >= distance) %>%  # Filter to keep only POIs at least 5 meters from boundaries
  select(1:2) %>%
  rename(xcoord_m = 1, ycoord_m = 2) %>%
  mutate(id = row_number()) %>%
  st_as_sf(coords = c("xcoord_m", "ycoord_m"), crs = 32739)

# Intersect with original polygons to extract attributes
polesInaccess <- st_intersection(poles_sf, polygons_map_sf)

# Add coordinates in both EPSG:32739 and EPSG:4326
polesInaccess <- cbind(polesInaccess,
                       st_coordinates(polesInaccess),
                       st_coordinates(st_transform(polesInaccess, 4326)))
names(polesInaccess)[names(polesInaccess) == "X"] <- "xcoord_m"
names(polesInaccess)[names(polesInaccess) == "Y"] <- "ycoord_m"
names(polesInaccess)[names(polesInaccess) == "X.1"] <- "Longitude"
names(polesInaccess)[names(polesInaccess) == "Y.1"] <- "Latitude"

# Prepare final dataset
polesInaccess$Date = "2022"
polesInaccess$Source = "PHOTO-INTERPRETATION"
polesInaccess = polesInaccess[, c("id", "Date", "Source", "xcoord_m", "ycoord_m", "Longitude",
 ↪  "Latitude", "Hab_L1", "Hab_L2", "Hab_L3", "Hab_L4", "geometry")]

# Save results
st_write(polesInaccess, paste0(save_path, "/", "Corrected_observed_map_", District, "_", Island,
 ↪  "_", Year1, "_Poles_of_inaccessibility_EPSG32739.shp"), driver = 'ESRI Shapefile', append =
 ↪  FALSE)
write.table(polesInaccess, file = paste0(save_path, "/", "Corrected_observed_map_", District, "_",
 ↪  Island, "_", Year1, "_Poles_of_inaccessibility_EPSG32739.csv"), sep = ";", dec = ".", row.names
 ↪  = FALSE)
```

2.7 Get Points on Surface

The `st_point_on_surface()` function generates a point within the surface of a given spatial geometry, such as a polygon from `polygons_map_sf` Note that the generated point will not be random; it will always be located within the polygon's surface and will be determined in a reproducible manner. Since the polygons have random shapes, a point near the center will be retrieved.

A unique identifier `id` is assigned to each point, starting from the maximum identifier of the previously computed poles of inaccessibility. Next,coordinates are added in both EPSG:32739 and EPSG:4326 formats, renaming the coordinate columns accordingly. It prepares the final dataset by including additional metadata such as the date and source of the data.

```r
# Compute internal points (centroids)
ptOnSurface <- polygons_map_sf %>%
    st_point_on_surface() %>%
    mutate(id = max(polesInaccess$id) + row_number())

# Add coordinates in both EPSG:32739 and EPSG:4326
ptOnSurface <- cbind(ptOnSurface, st_coordinates(ptOnSurface),
 ↪  st_coordinates(st_transform(ptOnSurface,
    4326)))
names(ptOnSurface)[names(ptOnSurface) == "X"] <- "xcoord_m"
names(ptOnSurface)[names(ptOnSurface) == "Y"] <- "ycoord_m"
names(ptOnSurface)[names(ptOnSurface) == "X.1"] <- "Longitude"
names(ptOnSurface)[names(ptOnSurface) == "Y.1"] <- "Latitude"

# Prepare final dataset
ptOnSurface$Date = "2022"
ptOnSurface$Source = "PHOTO-INTERPRETATION"
ptOnSurface = ptOnSurface[, c("id", "Date", "Source", "xcoord_m", "ycoord_m", "Longitude",
    "Latitude", "Hab_L1", "Hab_L2", "Hab_L3", "Hab_L4", "geometry")]

# Save results
st_write(ptOnSurface, paste0(save_path, "/", "Corrected_observed_map_", District,
    "_", Island, "_", Year1, "_Pts_on_surface_EPSG32739.shp"), driver = "ESRI Shapefile",
    append = FALSE)
write.table(ptOnSurface, file = paste0(save_path, "/", "Corrected_observed_map_",
    District, "_", Island, "_", Year1, "_Pts_on_surface_EPSG32739.csv"), sep = ";",
    dec = ".", row.names = FALSE)
```

2.8 Get random points

Dans cet exemple, le seuil de surface minimale des grands polygones est fixé à 2000m^{2}n le nombre de points aléatoires pouvant être créés étant fixé à 8, distants d'une distance de 12 m. **set.seed()** est utilisé pour fixer la graine du générateur de nombres aléatoires dans R à une valeur spécifique, dans ce cas, 3024. Cela garantit la reproductibilité lors de la génération de nombres aléatoires.

The script computes coordinates for additional random points within large polygons. To achieve this :

- It first identifies polygons with an area greater than a defined surface (`Area_min`)
- A random seed is set (`set.seed()`) for reproducibility, and parameters for generating random points are defined, including the number of points per polygon and the minimum distance between them.
- For each large polygon, the script generates random points using the `st_sample()` function and plots the polygon along with the generated points. It creates a spatial object for the random points, including relevant attributes from the polygon. The coordinates are computed in both EPSG:32739 and EPSG:4326 formats.
- All generated points are merged into a single spatial object, and a unique identifier `id` is assigned to each point.
- The final dataset is prepared with additional metadata, including the date and source.

```r
# Computing coordinates from random points
print("Computing coordinates for additional random points inside large polygons")


# Identify large polygons
Area_min = 2000
big_polygons_map_sf = subset(polygons_map_sf, polygons_map_sf$Surface >= Area_min)
print(paste0("Il y a ", nrow(big_polygons_map_sf), " polygones qui font plus de ",
    Area_min, " m2"))

# Fix random seed for reproducibility
set.seed(3024)


# Set parameters
n_random <- 8  # Number of random points per polygon
distance <- 12  # Minimum distance (m) between random points

# List to store generated points
points_within_polygons <- list()

# Generate random points within each large polygon
for (p in 1:nrow(big_polygons_map_sf)) {

    # Select the current polygon
    each_polygon <- big_polygons_map_sf[p, ]

    # Plot the polygon before generating points
    plot(each_polygon$geometry, col = "lightblue", main = paste("Polygon ID:", each_polygon$id))

    # Generate random points within the polygon
    random_points <- st_sample(each_polygon, size = n_random, exact = TRUE)

    # Check if any points were generated
    if (length(random_points) > 0) {
        # Plot the generated random points inside the polygon
        plot(random_points, col = "red", add = TRUE, pch = 20)  # Red points on top of the polygon

        # Create an sf object with random points and attributes
        random_points_sf <- st_sf(id = rep(each_polygon$id, each = length(random_points)),
            Nom_site = rep(each_polygon$Nom_site, each = length(random_points)),
            Surface = rep(each_polygon$Surface, each = length(random_points)), Hab_L1 =
             ↪  rep(each_polygon$Hab_L1,
                each = length(random_points)), Hab_L2 = rep(each_polygon$Hab_L2,
                each = length(random_points)), Hab_L3 = rep(each_polygon$Hab_L3,
                each = length(random_points)), Hab_L4 = rep(each_polygon$Hab_L4,
                each = length(random_points)), geometry = random_points  # Set geometry column
)

        # Compute coordinates in EPSG:32739 (meters)
        coords_32739 <- st_coordinates(random_points_sf)
        colnames(coords_32739) <- c("xcoord_m", "ycoord_m")

        # Transform to EPSG:4326 (Latitude & Longitude)
        random_points_sf <- st_transform(random_points_sf, crs = 4326)
        coords_4326 <- st_coordinates(random_points_sf)
        colnames(coords_4326) <- c("Longitude", "Latitude")

        # Append coordinate columns to the points dataset
        random_points_sf <- cbind(random_points_sf, coords_32739, coords_4326)

        # Store the results in the list
        points_within_polygons[[p]] <- random_points_sf
    }
}

# Merge all generated points into a single sf object
RandomPts <- do.call(rbind, points_within_polygons)

# Assign a new unique ID to each generated point
RandomPts$id <- (max(ptOnSurface$id, na.rm = TRUE) + 1):(max(ptOnSurface$id, na.rm = TRUE) +
    nrow(RandomPts))

# Transform to EPSG:32739 (xcoord_m nd ycoord_m)
RandomPts <- st_transform(RandomPts, crs = 32739)

# Prepare final file
RandomPts$Date = "2022"
RandomPts$Source = "PHOTO-INTERPRETATION"
RandomPts = RandomPts[, c("id", "Date", "Source", "xcoord_m", "ycoord_m", "Longitude",
    "Latitude", "Hab_L1", "Hab_L2", "Hab_L3", "Hab_L4", "geometry")]


# Save as Shapefile
st_write(RandomPts, paste0(save_path, "/", "Corrected_observed_map_", District, "_",
    Island, "_", Year1, "_Random_points_EPSG32739.shp"), driver = "ESRI Shapefile",
    append = FALSE)
write.table(RandomPts, file = paste0(save_path, "/", "Corrected_observed_map_", District,
    "_", Island, "_", Year1, "_Random_points_EPSG32739.csv"), sep = ";", dec = ".",
    row.names = FALSE)
```

This script facilitates the creation of three types of geolocated points, which are saved as both an ESRI Shapefile and a CSV file in a specified directory.