

```

# =====
# SCRIPT : CARTE TRAJETS + CLUSTERS (spots) POUR DONNEES ARGOS
#
# AUTHOR : Raphaël Musseau – BioSphère Environnement
# YEAR : 2025
# LICENSE: MIT
#
# DESCRIPTION
# Ce script produit une carte synthétique à partir des sorties du clustering :
# 1) Trajectoires (lignes) par individu, avec ruptures visuelles quand les fixes
#    sont trop espacés dans le temps ou trop éloignés dans l'espace.
# 2) Points ARGOS affichés avec un symbole dépendant de la classe de localisation (LC).
# 3) Polygones de "spots" (zones d'escale / hivernage) issus du clustering, tracés
#    en contours noirs au-dessus de tout le reste.
#
# DONNEES D'ENTRÉE (fichiers attendus)
# - argos_spots_points.csv :
#   * CSV des points annotés (sortie du script de clustering)
#   * Colonnes attendues au minimum : id, lon, lat, date_loc, lc
#   * (Peut contenir d'autres colonnes : spot_id, is_spot_point, etc.)
# - argos_spots_polygons.kml :
#   * KML des polygones de spots (convex hull) (sortie du script de clustering)
#
# PARAMÈTRES AJUSTABLES
# - gap_dt_h : rupture de trajectoire si l'intervalle de temps entre deux fixes
#              consécutifs dépasse ce seuil (heures)
# - gap_dist_km : rupture de trajectoire si la distance entre deux fixes consécutifs
#                dépasse ce seuil (km)
# - balise_labels : étiquettes lisibles pour la légende des individus
#
# FOND CARTOGRAPHIQUE
# - Fond Natural Earth (rnaturalearth) avec Maroc + Sahara occidental fusionnés
#   (pas de frontière interne) + ajout du trait de côte (ne_coastline).
# - Le cadre (xlim/ylim) est automatiquement calculé à partir de l'emprise des données.
#
# SORTIE
# - Une figure ggplot affichée à l'écran (print(p)).
#
# =====
suppressPackageStartupMessages({
  library(dplyr)
  library(ggplot2)
  library(sf)
  library(geosphere)
  library(rnaturalearth)
  library(rnaturalearthdata)
  library(ggspatial)
  library(scales)
})

# ---- 0) Vérifications -----

# Charger les fichiers les fichiers de sortie du script de clustering : d_clustered et spots_poly_sf
points_csv <- "C:/Users/Utilisateur/Desktop/RESULTS/argos_spots_points.csv"
poly_kml <- "C:/Users/Utilisateur/Desktop/RESULTS/argos_spots_polygons.kml"

if (!file.exists(points_csv)) stop("Fichier points introuvable : ", points_csv)
d_clustered <- read.csv(points_csv, stringsAsFactors = FALSE, check.names = FALSE)

# date_loc doit redevenir POSIXct (sinon difftime/arrange seront foireux)
if ("date_loc" %in% names(d_clustered) && !inherits(d_clustered$date_loc, "POSIXct")) {
  d_clustered$date_loc <- suppressPackageStartupMessages(as.POSIXct(
    as.character(d_clustered$date_loc), tz = "UTC",

```

```

    tryFormats = c("%Y-%m-%d %H:%M:%S", "%Y-%m-%d %H:%M",
                    "%d/%m/%Y %H:%M:%S", "%d/%m/%Y %H:%M")
  ))
}

# --- lecture du fichier KML de polygones (clusters) ---
spots_poly_sf <- NULL
if (file.exists(poly_kml)) {

  # 1) Lister les couches dispo
  lyr <- tryCatch(sf::st_layers(poly_kml), error = function(e) NULL)

  if (!is.null(lyr) && length(lyr$name) > 0) {

    # 2) Lire toutes les couches et empiler ce qui est polygonal
    polys_list <- list()

    for (L in lyr$name) {
      tmp <- tryCatch(sf::st_read(poly_kml, layer = L, quiet = TRUE), error = function(e) NULL)
      if (is.null(tmp) || nrow(tmp) == 0) next

      # Normaliser CRS si présent
      if (is.na(sf::st_crs(tmp))) sf::st_crs(tmp) <- 4326

      # Réparer puis tenter d'extraire/caster en polygones
      tmp <- tryCatch(sf::st_make_valid(tmp), error = function(e) tmp)

      # Si geometry collection/mixed, on cast vers MULTIPOLYGON
      tmp_poly <- tryCatch(sf::st_cast(tmp, "MULTIPOLYGON", warn = FALSE), error = function(e) NULL)

      if (!is.null(tmp_poly) && nrow(tmp_poly) > 0) {
        # Garder uniquement les polygones
        tmp_poly <- tmp_poly %>%
          dplyr::filter(sf::st_geometry_type(geometry) %in% c("POLYGON", "MULTIPOLYGON"))
        if (nrow(tmp_poly) > 0) polys_list[[length(polys_list) + 1]] <- tmp_poly
      } else {
        # fallback : filtrer direct si déjà polygonal
        tmp2 <- tmp %>%
          dplyr::filter(sf::st_geometry_type(geometry) %in% c("POLYGON", "MULTIPOLYGON"))
        if (nrow(tmp2) > 0) polys_list[[length(polys_list) + 1]] <- tmp2
      }
    }
  }

  if (length(polys_list) > 0) {
    spots_poly_sf <- do.call(rbind, polys_list)

    # CRS final en WGS84
    if (!is.na(sf::st_crs(spots_poly_sf)) && sf::st_crs(spots_poly_sf)$epsg != 4326) {
      spots_poly_sf <- sf::st_transform(spots_poly_sf, 4326)
    } else if (is.na(sf::st_crs(spots_poly_sf))) {
      sf::st_crs(spots_poly_sf) <- 4326
    }

    # Re-réparer au cas où
    spots_poly_sf <- tryCatch(sf::st_make_valid(spots_poly_sf), error = function(e) spots_poly_sf)
  } else {
    spots_poly_sf <- NULL
  }

} else {
  spots_poly_sf <- NULL
}
} else {

```

```

spots_poly_sf <- NULL
}

if (!exists("d_clustered")) stop("Objet 'd_clustered' introuvable : lance d'abord le script de nettoyage +
clustering.")
if (!exists("spots_poly_sf")) {
  warning("Objet 'spots_poly_sf' introuvable : aucun polygone de cluster ne sera dessiné.")
}

# paramètres pour les ruptures visuelles sur les trajectoires
gap_dt_h <- 12 # rupture si > 12 h entre deux fixes
gap_dist_km <- 150 # rupture si > 150 km entre deux fixes

# labels "prénoms" des balises (adapter si besoin)
balise_labels <- c(
  "283694" = "283694 (NAJAT)",
  "283695" = "283695 (INES)",
  "283696" = "283696 (NAYLE)"
)

# ---- 1) Préparation des données pour la carte -----

# On repart de d_clustered, on force id en facteur
d_plot <- d_clustered %>%
  filter(is.finite(lon), is.finite(lat)) %>%
  mutate(
    id_f = factor(as.character(id)), # important pour la couleur (discrète)
    lc = factor(lc, levels = c("1", "2", "3", "0", "A", "B", "C"))
  ) %>%
  arrange(id_f, date_loc)

# 1.1 Comptes par classe de localisation pour la légende
lc_show <- c("1", "2", "3", "0", "A", "B", "C")
tab_lc <- as.data.frame(table(factor(d_plot$lc, levels = lc_show)))
tab_lc <- tab_lc[tab_lc$Freq > 0, ]
lab_lc <- setNames(paste0(tab_lc$Var1, " (n=", tab_lc$Freq, ")"), tab_lc$Var1)

# On limite la légende aux classes les plus courantes
lc_breaks <- c("1", "2", "3", "0", "A", "B")
lc_breaks <- lc_breaks[lc_breaks %in% names(lab_lc)]

# 1.2 Segmentation des trajectoires pour les pointillés (gaps)
d_plot <- d_plot %>%
  group_by(id_f) %>%
  arrange(date_loc, .by_group = TRUE) %>%
  mutate(
    dt_h = as.numeric(difftime(date_loc, lag(date_loc), units = "hours")),
    ok = is.finite(lon) & is.finite(lat) & is.finite(lag(lon)) & is.finite(lag(lat)),
    dist_km = ifelse(
      ok,
      geosphere::distHaversine(cbind(lon, lat), cbind(lag(lon), lag(lat))) / 1000,
      NA_real_
    ),
    is_break = is.na(dt_h) | dt_h > gap_dt_h | dist_km > gap_dist_km,
    seg_id = cumsum(replace(is_break, is.na(is_break), FALSE))
  ) %>%
  ungroup() %>%
  select(-ok)

traj_gap <- d_plot %>%
  group_by(id_f) %>%
  arrange(date_loc, .by_group = TRUE) %>%
  mutate(

```

```

lon_lag = lag(lon),
lat_lag = lag(lat)
) %>%
ungroup() %>%
filter(is_break & is.finite(lon_lag) & is.finite(lat_lag))

traj_cont <- d_plot %>%
  filter(!is_break)

# ---- 2) Fond cartographique (Natural Earth, Maroc+Sahara union) ----

world <- rnaturalearth::ne_countries(scale = "medium", returnclass = "sf")

maroc_claimed <- world %>%
  dplyr::filter(admin %in% c("Morocco", "Western Sahara")) %>%
  dplyr::mutate(admin = "Maroc") %>%
  dplyr::summarise(geometry = sf::st_union(geometry), .groups = "drop")

world_custom <- world %>%
  dplyr::filter(!(admin %in% c("Morocco", "Western Sahara"))) %>%
  dplyr::bind_rows(maroc_claimed)

coast <- rnaturalearth::ne_coastline(scale = "medium", returnclass = "sf")

# Cadre de la carte
xpad <- max(diff(range(d_plot$lon, na.rm = TRUE))) * 0.06, 0.3)
ypad <- max(diff(range(d_plot$lat, na.rm = TRUE))) * 0.06, 0.3)
xlim_vals <- range(d_plot$lon, na.rm = TRUE) + c(-xpad, xpad)
ylim_vals <- range(d_plot$lat, na.rm = TRUE) + c(-ypad, ypad)

# ---- 3) Couleurs balises et préparation des clusters -----

# Palette par individu
pal_id <- scales::hue_pal()(length(levels(d_plot$id_f)))
names(pal_id) <- levels(d_plot$id_f)

balise_labels_present <- balise_labels[names(pal_id)]
balise_labels_present[is.na(balise_labels_present)] <- names(balise_labels_present)

# spots_poly_sf peut être NULL si aucun spot valide
has_spots <- (exists("spots_poly_sf") &&
  !is.null(spots_poly_sf) &&
  nrow(spots_poly_sf) > 0)

if (has_spots) {
  # On s'assure du bon CRS
  if (!is.na(sf::st_crs(spots_poly_sf)) && sf::st_crs(spots_poly_sf)$epsg != 4326) {
    spots_poly_sf <- sf::st_transform(spots_poly_sf, 4326)
  } else if (is.na(sf::st_crs(spots_poly_sf))) {
    sf::st_crs(spots_poly_sf) <- 4326
  }
}

# ---- 4) CARTE -----

p <- ggplot() +
  # Fond pays + côtes
  geom_sf(data = world_custom,
    fill = "grey95",
    color = "grey70",
    linewidth = 0.2) +
  geom_sf(data = coast,
    color = "grey60",

```

```

linewidth = 0.2) +

# Trajectoires continues
geom_path(data = traj_cont,
  aes(x = lon, y = lat,
    group = interaction(id_f, seg_id),
    colour = id_f),
  linewidth = 0.9,
  alpha = 0.9) +

# Ruptures (gaps) en pointillés
geom_segment(data = traj_gap,
  aes(x = lon_lag, y = lat_lag,
    xend = lon, yend = lat,
    colour = id_f),
  linewidth = 0.8,
  linetype = "22",
  alpha = 0.9) +

# Points
geom_point(data = d_plot,
  aes(x = lon, y = lat,
    colour = id_f,
    shape = lc),
  size = 2.2,
  stroke = 0.2) +

# Polygones de clusters : contours uniquement, AU-DESSUS de tout
{ if (has_spots)
  geom_sf(data = spots_poly_sf,
    aes(linetype = "Zone d'escale / hivernage"),
    fill = NA,
    color = "black",
    linewidth = 0.7,
    show.legend = TRUE)
else
  NULL } +

coord_sf(xlim = xlim_vals, ylim = ylim_vals, expand = FALSE) +

# Légende 1 : balises (couleurs)
scale_colour_manual(
  name = NULL,
  values = pal_id,
  labels = balise_labels_present,
  guide = guide_legend(nrow = length(pal_id),
    byrow = TRUE,
    order = 1)
) +

# Légende 2 : classes de localisation
scale_shape_manual(
  name = "Classes de localisation\n",
  values = c("1" = 16, "2" = 17, "3" = 15,
    "0" = 3, "A" = 8, "B" = 13, "C" = 7),
  breaks = lc_breaks,
  labels = lab_lc[lc_breaks],
  guide = guide_legend(order = 2,
    nrow = 3,
    byrow = TRUE,
    title.position = "top")
) +

```

```

# Légende 3 : clusters (linetype)
{ if (has_spots)
  scale_linetype_manual(
    name = NULL,
    values = c("Zone d'escale / hivernage" = "solid"),
    guide = guide_legend(order = 3,
                          override.aes = list(color = "black"))
  )
else
  NULL } +

labs(x = "Longitude", y = "Latitude") +
theme_minimal(base_size = 9) +
theme(
  legend.position = "right",
  legend.box = "vertical",
  legend.justification = "top",
  legend.title = element_text(size = 10, hjust = 0),
  legend.text = element_text(size = 9),
  legend.key.size = unit(0.9, "lines"),
  legend.spacing.y = unit(2, "mm"),
  legend.margin = margin(4, 4, 4, 4, "pt")
) +
annotation_north_arrow(
  location = "tl",
  which_north = "true",
  style = north_arrow_fancy_orienteering,
  height = unit(1.2, "cm"),
  width = unit(1.2, "cm")
) +
annotation_scale(
  location = "br",
  width_hint = 0.3,
  text_cex = 0.7,
  style = "ticks",
  line_width = 0.2
)

print(p)

## FIN DU SCRIPT

```