

```

# =====
#
# SCRIPT : ARGOS_SPATIOTEMPORAL_CLUSTERING.R
#
# AUTHOR: Raphaël Musseau – BioSphère Environnement
# CONTACT : musseau.biosphere-environnement@orange.fr
# YEAR: 2025
# LICENSE: MIT
#
# OBJECTIF
# Détecter des "spots" (amas spatio-temporels) dans des données GPS / ARGOS
# afin d'identifier des zones d'escale migratoire et/ou d'hivernage.
#
# SORTIES (dans un dossier choisi via une fenêtre) :
# 1) argos_spots_points.csv      : points + annotations (segments, clusters, spot_id...)
# 2) argos_spots_summary.csv    : résumé des spots valides (centroïdes, dates, rayon...)
# 3) argos_spots_points.kml     : points en KML
# 4) argos_spots_centroids.kml  : centroïdes des spots en KML (si spots valides)
# 5) argos_spots_polygons.kml   : polygones (convex hull) des spots (si spots valides)
#
# ENTREE ATTENDUE
# Un CSV avec au minimum :
# - id      : identifiant individu
# - lon, lat : positions WGS84 (degrés)
# - date_loc : date/heure (POSIXct ou chaîne convertible)
#
# Indicateurs de qualité des données :
# - lc      : classe de localisation (3,2,1,0,A,B,C)
# - smaj_km : demi-grand axe de l'ellipse (km)
#
# FILTRAGE QUALITE
# - Si lc + smaj_km disponibles :
#   * on garde toujours LC 1/2/3
#   * pour LC 0/A/B/C : on garde uniquement si smaj_km <= ellipse_max_km
# - Si lc disponible mais pas smaj_km :
#   * on garde uniquement LC 1/2/3 (0/A/B/C rejetées car non contrôlables)
# - Si lc absent :
#   * aucun filtre qualité possible (cas GPS pur) → on garde tout
#
# CLUSTERING
# 1) Segmentation temporelle (T_gap_hours) par individu :
#   rupture si l'écart entre deux fixes consécutifs dépasse T_gap_hours.
# 2) Clustering séquentiel dans chaque segment :
#   un point rejoint le cluster courant si sa distance au centroïde courant <= R_max_km,
#   sinon on démarre un nouveau cluster (séquentiel, suivant le temps).
# 3) Fusion de clusters successifs proches (R_fusion_km) via distance des centroïdes.
#   → donc des clusters issus de segments séparés (dt > T_gap_hours) peuvent être fusionnés
#   s'ils sont proches spatialement.
# 4) Un "spot" est un cluster final qui respecte :
#   n_points >= N_min_points ET duration_days >= duration_min_days
#
# PARAMETRES A AJUSTER SELON TYPE ANALYSE
# - ellipse_max_km : <- seuil ellipse pour accepter 0/A/B/C (<= 6km)
# - T_gap_hours    : <- rupture temporelle (<= 72 heures)
# - R_max_km       : rayon max (km) pour rester dans le cluster (>= 40 km)
# - R_fusion_km    : distance max (km) pour fusionner deux clusters successifs (<= 20 km)
# - N_min_points   : nb min de points pour qu'un spot soit retenu (n = 2)
# - duration_min_days : durée min (jours) pour qu'un spot soit retenu (>= 24 heures)
#
# =====

suppressPackageStartupMessages({
  library(dplyr)
  library(lubridate)
  library(geosphere)

```

```
library(sf)
library(tcltk)
})
```

1) PARAMÈTRES UTILISATEUR ----

```
input_file <- "C:/Users/Utilisateur/Desktop/RESULTS/argos_clean.csv"
```

```
T_gap_hours    <- 72
R_max_km       <- 40
R_fusion_km    <- 20
N_min_points   <- 2
duration_min_days <- 1
```

```
ellipse_max_km <- 6
```

2) FONCTIONS UTILITAIRES ----

```
to_time <- function(x) suppressWarnings(as.POSIXct(
  as.character(x), 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")
))
```

```
to_num <- function(x) suppressWarnings(as.numeric(gsub(",", ".", as.character(x))))
```

```
safe_write_csv <- function(x, file) {
  tryCatch(
    {
      write.csv(x, file, row.names = FALSE, fileEncoding = "UTF-8")
      TRUE
    },
    error = function(e) {
      message("Erreur d'écriture de : ", file)
      message("→ Vérifie que le fichier n'est pas ouvert (Excel...).")
      message("Détail : ", conditionMessage(e))
      FALSE
    }
  )
}
```

```
cluster_segment <- function(df_seg, R_max_km) {
  n <- nrow(df_seg)
  if (n == 0) return(df_seg %>% mutate(cluster_seq = NA_integer_))
  if (n == 1) {
    df_seg$cluster_seq <- 1L
    return(df_seg)
  }
}
```

```
cluster_seq <- integer(n)
current_k <- 1L
cluster_seq[1] <- current_k
```

```
cx <- df_seg$lon[1]
cy <- df_seg$lat[1]
c_n <- 1L
```

```
for (i in 2:n) {
  this_lon <- df_seg$lon[i]
  this_lat <- df_seg$lat[i]
```

```
dist_km <- geosphere::distHaversine(c(cx, cy), c(this_lon, this_lat)) / 1000
```

```
if (!is.finite(dist_km)) {
  current_k <- current_k + 1L
```

```

    cluster_seq[i] <- current_k
    cx <- this_lon; cy <- this_lat; c_n <- 1L
  } else if (dist_km <= R_max_km) {
    cluster_seq[i] <- current_k
    c_n <- c_n + 1L
    cx <- (cx * (c_n - 1L) + this_lon) / c_n
    cy <- (cy * (c_n - 1L) + this_lat) / c_n
  } else {
    current_k <- current_k + 1L
    cluster_seq[i] <- current_k
    cx <- this_lon; cy <- this_lat; c_n <- 1L
  }
}

df_seg$cluster_seq <- cluster_seq
df_seg
}

stop_with_head <- function(df, cols, n = 30, msg = "STOP") {
  print(utils::head(df[, cols, drop = FALSE], n))
  stop(msg, call. = FALSE)
}

## 3) LECTURE DU CSV ET PRÉPARATION ----

if (!file.exists(input_file)) stop("Fichier d'entrée introuvable : ", input_file)

cat("\n=== DEBUG INPUT ===\n")
cat("input_file : ", input_file, "\n")
cat("normalizePath : ", normalizePath(input_file, winslash = "/", mustWork = TRUE), "\n")
cat("file.info(size bytes) : ", file.info(input_file)$size, "\n")
cat("file.info(mtime) : ", as.character(file.info(input_file)$mtime), "\n")

d_raw <- read.csv(input_file, stringsAsFactors = FALSE, check.names = FALSE)

required_cols <- c("id", "lon", "lat", "date_loc")
if (!all(required_cols %in% names(d_raw))) {
  stop("Colonnes manquantes dans le CSV. Il faut au moins : ",
    paste(required_cols, collapse = ", "))
}

cat("\n=== DEBUG COLONNES ===\n")
print(names(d_raw))
cat("\nColonnes contenant 'smaj' :\n")
print(grep("smaj", names(d_raw), ignore.case = TRUE, value = TRUE))

if (!inherits(d_raw$date_loc, "POSIXct")) d_raw$date_loc <- to_time(d_raw$date_loc)

has_lc <- "lc" %in% names(d_raw)
has_smaj <- "smaj_km" %in% names(d_raw)

cat("\n=== DEBUG FLAGS ===\n")
cat("has_lc =", has_lc, "\n")
cat("has_smaj =", has_smaj, "\n")

d <- d_raw %>%
  filter(is.finite(lon), is.finite(lat), !is.na(date_loc))

if (has_lc) d$lc <- toupper(trimws(as.character(d$lc)))
if (has_smaj) d$smaj_km <- to_num(d$smaj_km)

if (has_lc && has_smaj) {
  d <- d %>%
    filter(
      lc %in% c("1", "2", "3") |

```

```

    (lc %in% c("0","A","B","C") & is.finite(smaj_km) & smaj_km <= ellipse_max_km)
  )
} else if (has_lc && !has_smaj) {
  d <- d %>% filter(lc %in% c("1","2","3"))
} else {
  # pas de lc : rien
}

d <- d %>%
  arrange(id, date_loc) %>%
  mutate(row_id = dplyr::row_number())

cat("\n=== RÉSUMÉ FILTRAGE QUALITÉ (entrée clustering) ===\n")
cat("Nb points après filtre minimal+qualité :", nrow(d), "\n")
if (has_lc) {
  cat("Effectifs par LC :\n")
  print(table(d$lc, useNA = "ifany"))
}
if (has_lc && has_smaj) {
  cat("Résumé smaj_km (après filtre) :\n")
  print(summary(d$smaj_km))
}

if (has_lc && has_smaj) {
  bad <- d %>%
    filter(lc %in% c("0","A","B","C") & is.finite(smaj_km) & smaj_km > ellipse_max_km)

  cat("\n=== CHECK: violations ellipse (0/A/B/C) APRÈS filtre ===\n")
  cat("Nb violations :", nrow(bad), "\n")

  if (nrow(bad) > 0) {
    stop_with_head(
      bad,
      cols = c("id","date_loc","lc","smaj_km","lon","lat"),
      n = 30,
      msg = "STOP: le filtre ellipse n'a PAS été appliqué."
    )
  }
}

if (has_lc && has_smaj) {
  bad_in_source <- d_raw %>%
    mutate(
      lc = toupper(trimws(as.character(lc))),
      smaj_km = to_num(smaj_km)
    ) %>%
    filter(lc %in% c("0","A","B","C") & is.finite(smaj_km) & smaj_km > ellipse_max_km)

  cat("\n=== INFO: violations ellipse DANS LE FICHIER SOURCE (avant filtre) ===\n")
  cat("Nb violations dans d_raw :", nrow(bad_in_source), "\n")
  if (nrow(bad_in_source) > 0) {
    cat("Exemples (source) :\n")
    print(head(bad_in_source[, c("id","date_loc","lc","smaj_km","lon","lat")], 10))
  }
}

```

4) SEGMENTATION TEMPORELLE (T_gap) ----

```

d_seg <- d %>%
  arrange(id, date_loc) %>%
  group_by(id) %>%
  mutate(
    dt_h = as.numeric(difftime(date_loc, lag(date_loc), units = "hours")),
    segment = cumsum(ifelse(is.na(dt_h) | dt_h > T_gap_hours, 1L, 0L))
  ) %>%

```

```

ungroup()

cat("\n=== RÉSUMÉ DES SEGMENTS TEMPORELS (T_gap) ===\n")
print(table(d_seg$id, d_seg$segment))

## 5) CLUSTERING SPATIO-TEMPOREL SÉQUENTIEL ----

d_clustered <- d_seg %>%
  arrange(id, segment, date_loc) %>%
  group_by(id, segment) %>%
  group_modify(~ cluster_segment(.x, R_max_km = R_max_km)) %>%
  ungroup()

clusters_seq <- d_clustered %>%
  filter(!is.na(cluster_seq)) %>%
  group_by(id, segment, cluster_seq) %>%
  summarise(
    n_points = n(),
    start_time = min(date_loc),
    end_time = max(date_loc),
    duration_days = as.numeric(difftime(end_time, start_time, units = "days")),
    lon_centroid = mean(lon, na.rm = TRUE),
    lat_centroid = mean(lat, na.rm = TRUE),
    .groups = "drop"
  )

cat("\n=== APERÇU DES CLUSTERS SÉQUENTIELS (avant fusion) ===\n")
print(head(clusters_seq, 10))

## 6) FUSION DES CLUSTERS PROCHES (R_fusion) ----

if (nrow(clusters_seq) > 0) {

  clusters_merged <- clusters_seq %>%
    arrange(id, start_time) %>%
    group_by(id) %>%
    mutate(
      prev_lon = lag(lon_centroid),
      prev_lat = lag(lat_centroid),
      dist_prev_km = ifelse(
        is.na(prev_lon),
        NA_real_,
        geosphere::distHaversine(
          cbind(prev_lon, prev_lat),
          cbind(lon_centroid, lat_centroid)
        ) / 1000
      ),
      new_group = ifelse(is.na(dist_prev_km) | dist_prev_km > R_fusion_km, 1L, 0L),
      final_group_index = cumsum(new_group)
    ) %>%
    select(-prev_lon, -prev_lat, -new_group) %>%
    ungroup()

  clusters_final <- clusters_merged %>%
    group_by(id, final_group_index) %>%
    summarise(
      n_points = sum(n_points),
      start_time = min(start_time),
      end_time = max(end_time),
      duration_days = as.numeric(difftime(end_time, start_time, units = "days")),
      lon_centroid = sum(lon_centroid * .data$n_points) / sum(.data$n_points),
      lat_centroid = sum(lat_centroid * .data$n_points) / sum(.data$n_points),
      .groups = "drop"
    )
}

```

```

d_clustered <- d_clustered %>%
  left_join(
    clusters_merged %>% select(id, segment, cluster_seq, final_group_index),
    by = c("id", "segment", "cluster_seq")
  )

clusters_final <- clusters_final %>%
  rowwise() %>%
  mutate(
    radius_km = {
      this_id <- id
      this_group <- final_group_index
      pts <- d_clustered %>% filter(id == this_id, final_group_index == this_group)
      if (nrow(pts) == 0) {
        NA_real_
      } else {
        max(
          geosphere::distHaversine(
            cbind(lon_centroid, lat_centroid),
            cbind(pts$lon, pts$lat)
          ) / 1000,
          na.rm = TRUE
        )
      }
    }
  ) %>%
  ungroup() %>%
  mutate(spot_id = paste0("SPOT_", id, "_", final_group_index))

} else {
  clusters_final <- tibble()
  d_clustered$final_group_index <- NA_integer_
}

cat("\n=== APERÇU DES CLUSTERS FINAUX (avant filtre N_min/durée) ===\n")
print(head(clusters_final, 10))

## 7) FILTRE SUR N_min ET DURATION_MIN ----

if (nrow(clusters_final) > 0) {

  valid_spots <- clusters_final %>%
    filter(n_points >= N_min_points, duration_days >= duration_min_days)

  cat("\nNombre de spots valides (N_min, durée_min) : ", nrow(valid_spots), "\n")

  d_clustered <- d_clustered %>%
    left_join(valid_spots %>% select(id, final_group_index, spot_id),
      by = c("id", "final_group_index")) %>%
    mutate(is_spot_point = !is.na(spot_id))

} else {
  valid_spots <- tibble()
  d_clustered$spot_id <- NA_character_
  d_clustered$is_spot_point <- FALSE
}

cat("\nNombre de points assignés à un spot : ",
  sum(d_clustered$is_spot_point, na.rm = TRUE), "\n")

## 8) OBJETS SF (POINTS, CENTROÏDES, POLYGONES) ----

points_sf <- st_as_sf(
  d_clustered,
  coords = c("lon", "lat"),

```

```

    crs = 4326,
    remove = FALSE
  )

  if (nrow(valid_spots) > 0) {
    spots_sf <- st_as_sf(
      valid_spots,
      coords = c("lon_centroid", "lat_centroid"),
      crs = 4326,
      remove = FALSE
    )

    spots_poly_sf <- points_sf %>%
      filter(is_spot_point) %>%
      group_by(id, spot_id, final_group_index) %>%
      summarise(
        geometry = sf::st_convex_hull(sf::st_union(geometry)),
        .groups = "drop"
      ) %>%
      left_join(valid_spots, by = c("id", "final_group_index", "spot_id"))

  } else {
    spots_sf <- NULL
    spots_poly_sf <- NULL
  }

## 9) CHOIX DU DOSSIER DE SORTIE ----

cat("\nUne fenêtre va s'ouvrir pour choisir le dossier de sortie...\n")
dest_dir <- tcltk::tk_choose.dir(caption = "Choisissez un dossier pour enregistrer les fichiers")

if (is.na(dest_dir) || !nzchar(dest_dir)) stop("Aucun dossier choisi. Arrêt du script.")

dest_dir <- normalizePath(dest_dir, winslash = "/", mustWork = TRUE)

outfile_points <- file.path(dest_dir, "argos_spots_points.csv")
outfile_spots <- file.path(dest_dir, "argos_spots_summary.csv")
outfile_kml_pts <- file.path(dest_dir, "argos_spots_points.kml")
outfile_kml_cent <- file.path(dest_dir, "argos_spots_centroids.kml")
outfile_kml_poly <- file.path(dest_dir, "argos_spots_polygons.kml")

## 10) EXPORTS CSV + KML ----

ok_points <- safe_write_csv(d_clustered, outfile_points)
ok_spots <- safe_write_csv(valid_spots, outfile_spots)

st_write(points_sf, outfile_kml_pts,
  driver = "KML",
  layer = "TrackPoints",
  append = FALSE)

if (!is.null(spots_sf) && nrow(spots_sf) > 0) {
  st_write(spots_sf, outfile_kml_cent,
    driver = "KML",
    layer = "SpotsCentroids",
    append = FALSE)
}

if (!is.null(spots_poly_sf) && nrow(spots_poly_sf) > 0) {
  st_write(spots_poly_sf, outfile_kml_poly,
    driver = "KML",
    layer = "SpotsPolygons",
    append = FALSE)
}

```

```

cat("\n=== EXPORT TERMINÉ ===\n")
cat("Points + annotations  :", outfile_points, if (!ok_points) " (échec écriture)" else "", "\n")
cat("Résumé des spots      :", outfile_spots, if (!ok_spots) " (échec écriture)" else "", "\n")
cat("KML points            :", outfile_kml_pts, "\n")
if (!is.null(spots_sf) && nrow(spots_sf) > 0) {
  cat("KML centroïdes spots  :", outfile_kml_cent, "\n")
}
if (!is.null(spots_poly_sf) && nrow(spots_poly_sf) > 0) {
  cat("KML polygones spots   :", outfile_kml_poly, "\n\n")
} else {
  cat("Pas de spots valides → pas de KML de centroïdes/polygones.\n\n")
}

## FIN DU SCRIPT

```