3.3_Calculation_sea_distances.Rmd

Iris Van Paemel

2024-06-12

3.3_Calculation_sea_distances.R

Input

The input data consists of csv files: - Output/Species_Location.csv => header consists of "Specieslist", "BelgianCoast", "Getxo", "Koster", "Laeso", "Limfjord", "Plymouth", "Roscoff", "SwedishWestCoast", "TZS", "Vigo" - Inputs/Coordinates.csv => header consists of "Observatory.ID", "Latitude", "Longitude"

Output

The output data consists of csv files: - sea_distances/[species_name]*distancesTo*[Location_name].csv => header consists of "x", "year", "month", "country" - fly_distances/[species_name]*distancesTo*[Location_name].csv => header consists of "x", "year", "month", "country" - OccurrenceData_test/[species_name].csv => header consists of "Longitude", "Latitude", "year", "month", "country"

script + descriptions

- First install and load the packages
- Set the working directory and load the input data /! Replace by your data if you use other filenames/paths

```
##############################################################################
# This script is an alternative script for calculating sea distances and fly distances
##############################################################################

#############################################################################################
# LOAD PACKAGES
#############################################################################################
library("gdistance")
library("dplyr")
require("geosphere")
require("rgbif")
library("ggplot2")
library("tidyr")
library("worrms")
library("sf")
library("sp")
library("raster")
library("rnaturalearth")
library("rnaturalearthdata")
```

```r
library("maps")
library("FRK")


################################################################################################
# LOAD DATA
################################################################################################
# Set working directory to directory where the R-script is saved
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
# Read a species list
df <- read.csv("Output/Species_Location.csv")
# Read coordinates file
Coordinates <- read.csv("Inputs/Coordinates.csv")
```

RESHAPE DF (line 38 in 3.3_Calculation_sea_distances.R script)

- The pivot_longer function is used to reshape df to long format, all columns in df except Specieslist will be pivoted. They will be treated as value columns
- Only the values that are bigger than 0 will be kept

```r
################################################################################################
# RESHAPE DF WITH LOCATIONS AND SPECIES FROM WIDE TO LONG FORMAT
################################################################################################

long <- pivot_longer(df, !Specieslist)
# from tidyr package, reshape df from wide to long format
long <- long[long$value > 0, ]
```

FETCH OCCURRENCE DATA (line 50 in 3.3_Calculation_sea_distances.R script)

- Ensure_columns function:
  - uses df and required columns
  - setdiff is used to find which columns are present in required_columns but not in colnames(df)
  - the for loop will check if the column in missing columns doesn't exist, and then makes this column and adds NA values to it
- Fetch_data_in_batches function:
  - set start to 0 = start in list of records in GBIF
  - start combined_data dataframe
  - fetch records from GBIF with settings
  - put res$data into the combined_data dataframe
- assign the required columns
- start a for loop that iterates over species names in the long dataframe
  - print the species name
  - put the filename into the variable 'occurrence_coord' /! Replace by your data if you use other filenames/paths
  - check if the file exists, if so, read the file
  - else: start fetching the data by using the function fetch_data_in_batches
  - Initialize an empty list to store processed data frames
  - Loop over each data frame in the list, ensure columns, and select required columns
  - check if temp_df is not NULL and not empty

    - Ensure the required columns are present
    - else: print that the dataframe is empty.

- check if length of processed_data is bigger than zero: batches of data
- merge the lists of different batches together into res_total
- check the sum of NA values in res_total
- else: print an error message
- rename the columns
- remove occurrences where longitude or latitude is NA
- If the amount of rows or res_total = 0 then:
    - check if the directory test_outputs/errors exists
    - if not: create one and write the error file
    - if it does exist: write the error file
- Lastly: write the file for this species with res_total

```
##############################################################################################
# FIRST CHECK IF FILE WITH OCCURRENCE DATA IN OCCURRENCEDATA DIRECTORY EXISTS
# IF NOT: MAKE ONE AND GET DATA FROM GBIF
# limit in occ_data() function is changeable for personal preference
##############################################################################################

# Function to ensure all required columns are present (used in coming for loop)
ensure_columns <- function(df, required_columns) {
  missing_columns <- setdiff(required_columns, colnames(df))
  for (col in missing_columns) {
    if (!col %in% colnames(df)) {
      df[[col]] <- NA
    }
  }
  return(df)
}
fetch_data_in_batches <- function(species_name, basisOfRecord, batch_size = 10000) {
  start <- 0
  combined_data <- data.frame()
    res <- occ_data(scientificName = species_name,
                    hasCoordinate = TRUE,
                    limit = batch_size,
                    basisOfRecord = basisOfRecord,
                    continent = "europe")
  combined_data <- rbind(combined_data, res$data)
  Sys.sleep(1)  # Adding a small delay to be polite to the server

  return(combined_data)
}

required_columns <- c('decimalLongitude', 'decimalLatitude', 'year', 'month', 'country')

for (species_name in unique(long$Specieslist)){
  print(paste0("species_name: ", species_name))
  # if file exists:        put content into variable res
  # if file doesn't exist: make one, get data, and put into variable res
  occurrence_coord <- paste0("OccurrenceData_test/", species_name, ".csv")
  if (file.exists(occurrence_coord) == TRUE) {
```

```
  res <- read.csv(occurrence_coord, header = TRUE)
} else {
  data_list <- list(
    fetch_data_in_batches(species_name, "Observation"),
    fetch_data_in_batches(species_name, "Machine observation"),
    fetch_data_in_batches(species_name, "Human observation"),
    fetch_data_in_batches(species_name, "Material sample"),
    fetch_data_in_batches(species_name, "Living specimen"),
    fetch_data_in_batches(species_name, "Occurrence"))

  # Initialize an empty list to store processed data frames
  processed_data <- list()

  # Loop over each data frame in the list, ensure columns, and select required columns
  for (i in seq_along(data_list)) {
    temp_df <- data_list[[i]]

    if (!is.null(temp_df) && nrow(temp_df) > 0) { # check if temp_df is not NULL and not empty
      # Ensure the required columns are present
      temp_df <- ensure_columns(temp_df, required_columns)
      temp_df <- temp_df[, required_columns]
      processed_data[[i]] <- temp_df

    } else {
      print(paste0("Data frame ", i, " is empty."))
    }
  }
}

if (length(processed_data) > 0) {
  res_total <- do.call(rbind, processed_data)
  print(paste0("Total NA values in res_total: ", sum(is.na(res_total))))
} else {
  print(paste0("No data to combine for species: ", species_name))
}
#rename the column names
colnames(res_total) <- c('Longitude', 'Latitude', 'year', 'month', 'country')
# Remove occurrences where longitude or latitude is NA
res_total <- res_total[!is.na(res_total$Latitude) & !is.na(res_total$Longitude),]
# check if there's no information for a species
if (nrow(res_total) == 0) {
  error_message <- paste0("No information found on GBIF for ", species_name)

  # check if directory with error messages exists, if it doesn't: make one
  if (!dir.exists("test_outputs/errors")){
    dir.create("test_outputs/errors", recursive = TRUE)
    error_file_name <- paste0("test_outputs/errors/error_", species_name, ".csv")
    # error files written to test_outputs/errors/
    write.csv(error_message, file = error_file_name)
    return(FALSE)

    # write error file to the directory
  } else {
```

```
      error_file_name <- paste0("test_outputs/errors/error_", species_name, ".csv")
      # error files written to test_outputs/errors/
      write.csv(error_message, file = error_file_name)
      return(FALSE)
    }
  }
  print("file has successfully been written")
  write.csv(res_total, file = occurrence_coord)
}
```

FIRST PART OF DISTANCE CALCULATION FUNCTION (line 150 in 3.3_Calculation_sea_distances.R script) = first part of the function: don't run this part in this Rmd, because the function is not closed

- Iterate over species_name and location_name
- Initialize an empty list to store error messages = error_messages
- Define a helper function to add error messages
- Print species name and location
- Try to read occurrence data in a tryCatch function
  - put the file path into the variable 'occurrence_coord' /! Replace by your data if you use other filenames/paths
  - read the file
  - add an error message when this doesn't work
- if the OccurrenceData variable is null, add an error message and return NA as a result
- Try to get coordinates for ARMS location
- use grep to get the row from the Coordinates df where location_name is present
- if the row index is zero: add an error message
- save longitude and latitude for the row that you selected with grep
- make a dataframe out of the longitude and latitude called samplelocation
- check if OccurrenceData has coordinates

```
################################################################################################
# REVISION: CALCULATE DISTANCES
################################################################################################

# Iterate over species_name and location_name
Calculation_seadistance <- function(species_name, species_location){
  # Initialize an empty list to store error messages
  error_messages <- list()

  # Define a helper function to add error messages
  add_error_message <- function(message) {
    error_messages <<- c(error_messages, message)
  }
  # Print species name and location
  print(paste0("species_name: ", species_name))
  print(paste0("species_location: ", species_location))

  # Try to read occurrence data
  OccurrenceData <- tryCatch({
    occurrence_coord <- paste0("OccurrenceData_test/", species_name, ".csv")
    read.csv(occurrence_coord, header = TRUE)
  }, error = function(e) {
```

```
  add_error_message(paste("Error reading occurrence data for", species_name, ":", e$message))
  return(NULL)
})

if (is.null(OccurrenceData)) return(list(result = NA, error_messages = error_messages))

# Try to get coordinates for ARMS location
# use grep to get the row from the Coordinates df where location_name is present
location_row_index <- grep(species_location, Coordinates$Observatory.ID)
if (length(location_row_index) == 0) {
  add_error_message(paste("Location not found in Coordinates for", species_location))
  return(list(result = NA, error_messages = error_messages))
}
# save longitude and latitude for the row that you selected with grep
longitude <- Coordinates[location_row_index, "Longitude"]
latitude <- Coordinates[location_row_index, "Latitude"]
# make a dataframe out of the longitude and latitude called samplelocation
samplelocation <- data.frame(Latitude = latitude, Longitude = longitude)

# check if OccurrenceData has coordinates
if (nrow(OccurrenceData) < 1) {
  add_error_message("OccurrenceData has no coordinates")
  return(list(result = NA, error_messages = error_messages))
}
```

SECOND PART OF DISTANCE CALCULATION FUNCTION (line 197 in 3.3_Calculation_sea_distances.R script) = don't run this part in this Rmd file, because it is only a part of a function

- Load medium scale natural earth countries as an sf (simple features) object
- Create a raster object with a global extent and resolution of 0.1 degrees
- Rasterize the 'world' sf object, assigning a value of 1 to cells with country presence
- Reclassify the raster: convert all values of 1 to Inf (infinity)
- Replace NA values in the 'costs' raster with 1
- Initialize lists to store distances
- start a for loop to iterate over OccurrenceData
- print for which latitude and longitude of the occurrenceData the distance will be calculated to the samplelocation latitude and longitude
- start a tryCatch for the calculation of the sea distance
- check if a transition_matrix exists
- create a transition object for adjacent cells
- Set infinite costs to NA to prevent travel through these cells
- Save transition matrix
- if transitMatrix exists, load it
- Define points using correct projection
- Check if the OccurrenceData point is on land, if so, skip this and put inf as a result
- make an error message when the point is on land
- Coerce points to SpatialPointsDataFrame for compatibility with gdistance
- Compute cost distance
- Calculate the shortest path
- plot the shortest distance on a map, when uncommented
- ensure CRS is set on the original SpatialLines object
- set it if it is not set.
- Convert SpatialLines to sf object

- Confirm CRS is set for sf object, if not, set it
- Transform to a suitable projected CRS for distance calculation
- Calculate the length in meters
- Print the length
- add the length to the sea_distances variable
- define the error function in the tryCatch function
- add an error message when the calculation didn't work
- The iteration over OccurrenceData stops here
- start a for loop to iterate over OccurrenceData again for fly distance calculation (this is necessary to have the same amount of INF values in both sea and fly distances)
- Define points using correct projection
- Check if the OccurrenceData point is on land
- If the point is on land, add an error message and add 'Inf' to the flying_distances list

```
#############################################################################
# DISTANCES CALCULATION
#############################################################################

# Load world data and prepare the raster
world <- ne_countries(scale = "medium", returnclass = "sf") # Load medium scale natural earth countri
r <- raster(extent(-180, 180, -90, 90), res = 0.1) # Create a raster object with a global extent and
r <- rasterize(world, r, field = 1, fun = max, na.rm = TRUE) # Rasterize the 'world' sf object, assig
costs <- reclassify(r, cbind(1, Inf)) # Reclassify the raster: convert all values of 1 to Inf (infini
costs[is.na(costs)] <- 1  # Replace NA values in the 'costs' raster with 1

# Initialize lists to store distances
sea_distances <- c()
flying_distances <- c()

# for loop to iterate over OccurrenceData
for (row in 1:nrow(OccurrenceData)) {
  print(paste0("Calculating latitude: ", OccurrenceData[row, 3], " and longitude: ", OccurrenceData[r
  print(paste0("for samplelocation latitude, longitude: ", samplelocation[,1], " ", samplelocation[,2

  ##################
  ## SEA DISTANCE##
  ##################

  sea_distance <- tryCatch({

    transition_matrix <- "transitMatrix.rds"
    if (!file.exists(transition_matrix)) {
      # Create a transition object for adjacent cells
      transitMatrix <- transition(costs, transitionFunction = function(x) 1/mean(x), directions = 16)
      # Set infinite costs to NA to prevent travel through these cells
      transitMatrix <- geoCorrection(transitMatrix, scl = TRUE)
      # Save/Load transition matrix
      saveRDS(transitMatrix, file = "transitMatrix.rds")

    } else {
      transitMatrix <- readRDS(file = "transitMatrix.rds")
    }

    # Define points using correct projection
```

```
    point1 <- SpatialPoints(cbind(samplelocation$Longitude, samplelocation$Latitude), proj4string = CR
    point2 <- SpatialPoints(cbind(OccurrenceData[row, 2], OccurrenceData[row, 3]), proj4string = CRS(p

    # Check if the OccurrenceData point is on land, if so, skip this and put inf as a result
    if (!is.na(raster::extract(r, point2))) {
      add_error_message(paste("Point on land detected for", species_name, "at", OccurrenceData[row, 2
      sea_distances <- append(sea_distances, Inf)
      next
    }

    # Coerce points to SpatialPointsDataFrame for compatibility with gdistance
    point1_df <- SpatialPointsDataFrame(coords = point1, data = data.frame(id = 1), proj4string = CRS
    point2_df <- SpatialPointsDataFrame(coords = point2, data = data.frame(id = 2), proj4string = CRS
    # Compute cost distance
    cost_distance <- costDistance(transitMatrix, point1_df, point2_df)
    # Calculate the shortest path
    shortest_path <- shortestPath(transitMatrix, point1_df, point2_df, output = "SpatialLines")

    # Plotting the shortest path and the world map
    #plot(r, main = "Shortest Water Path")
    #plot(world, add = TRUE, col = "grey")
    #plot(shortest_path, add = TRUE, col = "blue", lwd = 2)
    #points(point1_df, col = "red", pch = 20)
    #points(point2_df, col = "green", pch = 20)

    # Assuming 'shortest_path' is your SpatialLines object from the shortestPath function
    # First, ensure the CRS is set on the original SpatialLines object
    crs_info <- proj4string(shortest_path)  # or use crs(shortest_path) if using `sp`

    # If it's not set, set it here, assuming the original data was in WGS 84 (EPSG:4326)
    crs_info <- proj4string(shortest_path)
    if (is.na(crs_info)) {
      proj4string(shortest_path) <- CRS("+init=epsg:4326")
    }

    # Convert SpatialLines to sf object
    shortest_path_sf <- st_as_sf(shortest_path)

    # Confirm CRS is set for sf object, if not, set it:
    if (is.na(st_crs(shortest_path_sf))) {
      st_crs(shortest_path_sf) <- 4326  # EPSG code for WGS 84
    }

    # Transform to a suitable projected CRS for distance calculation (e.g., UTM zone 33N)
    shortest_path_utm <- st_transform(shortest_path_sf, 32633)  # UTM zone 33N

    # Calculate the length in meters
    path_length <- st_length(shortest_path_utm)

    # Print the length
    print(paste0("distance through sea in m: ", path_length))
    sea_distances <- append(sea_distances, path_length)
```

```
    }, error = function(e) {
      add_error_message(paste("An error occurred during sea distance calculation for", species_name, "i
      sea_distances <- append(sea_distances, NA)
      return(NULL)

    }) # trycatch() closed
  } # iteration over OccurrenceData stopped

  # for loop to iterate again over OccurrenceData for fly distances
  for (row in 1:nrow(OccurrenceData)) {

    # for this calculation, longitude comes first and then latitude!!!
    # Define points using correct projection
    point1 <- SpatialPoints(cbind(samplelocation$Longitude, samplelocation$Latitude), proj4string = CRS
    point2 <- SpatialPoints(cbind(OccurrenceData[row, 2], OccurrenceData[row, 3]), proj4string = CRS(pr

    # Check if the OccurrenceData point is on land
    if (!is.na(raster::extract(r, point2))) {
      add_error_message(paste("Point on land detected for", species_name, "at", OccurrenceData[row, 2],
      flying_distances <- append(flying_distances, Inf)
      next
    }
```

THIRD PART OF DISTANCE CALCULATION FUNCTION (line 323 in 3.3_Calculation_sea_distances.R script) = don't run this part in this Rmd file, because it is only a part of a function

- Calculate the straight-line distance (accounting for the Earth's curvature)
- print this distance
- append the distance to the flying_distances list
- stop the iteration over OccurrenceData
- create dataframes out of sea_distances and flying_distances using a function
- check if the length of the vectors are the same
- make the dataframe
- if the lengths are not the same: give an error message
- some checks are included to check the lengths of the vectors
- create sea and fly dataframes
- Define file paths
- Create directories if they do not exist
- Write data frames to CSV files
- Return the result and the list of error messages
- run the function, the output is saved in the variable 'results', this is mostly not necessary because the results are directly saved in files inside directories
- write the error file

```
#####################
    ## FLYING DISTANCE ##
    #####################

    # Calculate the straight-line distance (accounting for the Earth's curvature)
    straight_line_distance <- distHaversine(coordinates(point1), coordinates(point2))
    print(paste("Straight line distance:", straight_line_distance, "meters"))
    flying_distances <- append(flying_distances, straight_line_distance)
```

```r
  } # iteration over OccurrenceData stopped

  # Create data frame if lengths match
  create_data_frame <- function(distances, year, month, country) {
    if (length(distances) == length(year) && length(year) == length(month) && length(month) == length(co
      return(data.frame(
        x = distances,
        year = year,
        month = month,
        country = country
      ))
    } else {
      stop("Lengths of vectors do not match. Please ensure all vectors have the same length.")
    }
  }
  ### CHECKS IF NECESSARY ###
  #cat("Length of flying_distances: ", length(flying_distances), "\n")
  #cat("Length of sea_distances: ", length(sea_distances), "\n")
  #cat("Length of OccurrenceData$year: ", length(OccurrenceData$year), "\n")
  #cat("Length of OccurrenceData$month: ", length(OccurrenceData$month), "\n")
  #cat("Length of OccurrenceData$country: ", length(OccurrenceData$country), "\n")

  # Create sea data frame
  sea_data <- create_data_frame(sea_distances, OccurrenceData$year, OccurrenceData$month, OccurrenceData

  # Create fly data frame
  fly_data <- create_data_frame(flying_distances, OccurrenceData$year, OccurrenceData$month, OccurrenceD

  # Define file paths
  sea_distance_file <- paste0("test_outputs/sea_distances/", species_name, "_distancesTo_", species_loca
  fly_distance_file <- paste0("test_outputs/fly_distances/", species_name, "_distancesTo_", species_loca

  # Create directories if they do not exist
  if (!dir.exists("test_outputs/sea_distances")) {
    dir.create("test_outputs/sea_distances", recursive = TRUE)
  }

  if (!dir.exists("test_outputs/fly_distances")) {
    dir.create("test_outputs/fly_distances", recursive = TRUE)
  }

  # Write data frames to CSV files
  write.csv(sea_data, file = sea_distance_file, row.names = FALSE)
  write.csv(fly_data, file = fly_distance_file, row.names = FALSE)
  # Return the result and the list of error messages
  list(result = list(sea_distances = sea_distances, flying_distances = flying_distances), error_messages

}

results <- lapply(seq_len(nrow(long)), function(i) Calculation_seadistance(long$Specieslist[i], long$nam
```