Animals in space and time:
spatio–temporal movement pattern analysis

Doctoral thesis

at the Faculty of Biology, Chemistry and Geosciences,
University of Bayreuth
to attain the academic degree of Doctor of Natural Science (Dr. rer. nat.)

submitted by
Mirjana Bevanda

born in
Göppingen

Bayreuth, 2015

Appendix – R-Code

Chapter 1 - Migration pattern of red deer in the Bohemian Forest

```
###
### extract_local_slope.r
###
### extract slope of NDVI values for all location points
#############################################################################################################

#############################################################################################################
###     LIBRARIES                                                                                         ###
#############################################################################################################
library("stringr")
library("raster")
library("rgdal")
library("sp")
library("rgeos")
library("zoo")

#############################################################################################################
###     PATHS                                                                                             ###
#############################################################################################################
mainfolder <- "D:/WORK/DATA/01_GPS/GPS_DATA_HIRSCH/project_reddeer_NDVI"
datafolder <-  paste0(mainfolder, "/01_data/01_deer_data")
codefolder <- paste0(mainfolder, "/02_code")
### ndvi (calculated slope between two time steps)
ndvifolder <- paste0(mainfolder, "/01_data/02_NDVI/max_slope_ndvi_filtered")
ndviorigfolder<-paste0(mainfolder, "/01_data/02_NDVI/cropped_NDVI")
gatterfolder <- paste0(mainfolder, "/01_data/03_winter_enclosures")
imagefolder <- paste0(mainfolder, "/01_data/04_images")
sosfolder <- paste0(mainfolder, "/01_data/02_NDVI/start_of_season_tiffs")
asterfolder <- "D:/WORK/DATA/02_RS_GIS/GeodatenShare/ASTER/ASTERprocessing"
aster <- raster(paste0(asterfolder, "/crop_aster_utm.tif"))

#############################################################################################################
###     SETTINGS                                                                                          ###
#############################################################################################################
### data input (cleaned GPS data as data frame) - not restricted to winter enclosure dates
deerdatadf <- "deer_cleaned_and_random_sampled.RData"
### input shape file of winter enclosures
gatter <- readOGR(dsn=gatterfolder, layer="enclosures_BWCZ_UTM")
### settings for ndvi extraction
```

```
buffercols <- c("minslope", "maxslope", "meanslope", "medianslope", "sdslope")
### must correspond with buffercols
myextracts <- function(x){paste(min(x, na.rm=T),max(x, na.rm=T),mean(x, na.rm=T),
                                median(x, na.rm=T), sd(x, na.rm=T),sep=";" )}
### buffersize around location point of animal to calculate ndvi extraction - size in meter
buffersizes <- c(250, 500, 2000)

###     START     ##########################################################################
### load data animals (name of object = deer)
setwd(datafolder) ; load(deerdatadf)
deer <- deerrandomsamples
### add new columes for the results of buffer calculations and centerslope
### (centerslope = location point of animal)
allbuffercols <- c(unlist(lapply(buffersizes, function(b) str_c(buffercols, "_",b))), "centerslope")
#deer[,allbuffercols] <- NA
### GPS data ranges from 2002 - 2012. NDVI data only available until (including) 2011 -> exclude 2012
years <- unique(deer$yeardate)
years <- years[years !=2012]
### get S4 animal data
coordinates(deer)<-c("Easting", "Northing")
### project
proj4string(deer)<-proj4string(gatter)
#read in first ndvi file as reference
ndvis <- raster(list.files(ndvifolder, full.names=TRUE)[1])
#calculate the ID of the raster pixel the deer is in, will speed up calculation considerably
deer@data$centercellID<- cellFromXY(ndvis,deer)
### extract elevation
datasub <- deer
deer$elev <- extract(aster, datasub)

##########################################################################################
### extraction of desired values

valsallyears<-NULL
for(year in years){
    ### load raster for respective year
    ndvis <- raster(list.files(ndvifolder, pattern=str_c(year), full.names=TRUE))
    ### select animal data for respective year
    deeryear <- deer[deer$yeardate==year,]
    ### convert to S4
    #coordinates(deeryear)<-c("Easting", "Northing")
    #get all unique CellIDs with deer in them
```

```
  uniquedeer<-deeryear[!duplicated(deeryear@data$centercellID),]
  ### extraction for different buffersizes around location point
  valsallbuffer <- data.frame(centerslope=extract(ndvis,uniquedeer), yeardate=year,
                              centercellID=uniquedeer@data$centercellID)

  for(buffersize in buffersizes){
      #myextract returns all values as a string
      vals<-extract(ndvis, uniquedeer, buffer=buffersize, fun=myextracts)
      #remove NA Values can give warnings
      vals[which(is.na(vals))]<-myextracts(NA)
      vals2<-read.table(textConnection(vals), sep = ";")
      colnames(vals2) <- str_c(buffercols, "_", buffersize)
      if(dim(vals2)[1]!= length(uniquedeer)) stop("mismatch in Dimensions!")
      valsallbuffer <- cbind(valsallbuffer, vals2)
      print(str_c("Done Buffer ", buffersize))
    }
    valsallyears<-rbind(valsallyears,valsallbuffer)
    ### add values to data frame
    print(str_c("Done year ", year, " ....."))
  }
  print("Done Extracting, Combining Data")

  #join extracted data to points via year and centercellID
  deer@data<-merge(deer@data, valsallyears, by=c("yeardate", "centercellID"), all.x=TRUE)

  save(deer, file=str_c(datafolder, "/deer_extracted_buffer_ndvi_values.RData"))
  ### objectname = deer
  ##############################################################################
  # load(str_c(datafolder, "/deer_extracted_buffer_ndvi_values.RData"))

  ### extract if animals are in winter enclosures or not
  ### create id for winter enclosures, based on "gatter id"
  gattermin<-gatter[,"OBJECTID"]
  names(gattermin)<-c("GATTERID")
  ### query: point in polygon. NA for not in polygon = not in winter enclosure
  areinside<-over(deer, gattermin)
  ### add to data frame
  deer$gatterid <- areinside
  ### calculate julian date and modis time series day and add to data frame
  deer$juliandate <- as.POSIXlt(deer$Date)$yday+1
  deer$modisTS<-deer$juliandate/16+1

  deersub <- deer[,c("yeardate", "centercellID", "name", "year", "collar", "DateTime", "Date",
```

```
    "Ohrmarke", "Geschlecht", "Alter", "id", "IDO", "tdiff1", "weekofyear",
    "biweekofyear", "month", "row_ID", "timegroup", "centerslope", "gatterid",
    "juliandate", "modisTS")]

deersub@data$gatterid <- deersub@data$gatterid$GATTERID
### subset deer with enclosures in NPBW
### opning in April or May
deerenc <- subset(deersub, deersub$month %in% c("Apr", "May"))
deerenc@data <- droplevels(deerenc@data)
deerenc.spring <- subset(deerenc, deerenc$gatterid %in% c(14,15,16,17))
deerenc.spring@data <- droplevels(deerenc.spring@data)
head(deerenc.spring)
names.animals.in.enclosure <- levels(deerenc.spring$id)

#############################################################################
### real opening dates
### extract real opening dates of winter enclosures based on a series of 10 location points
### which are not located within winter enclosures
# setwd(codefolder)
# source("extract_opening_dates.r")
load(str_c(datafolder, "/deer_date_enclosuers_open.RData")) ### objectname = deer.opening
#############################################################################

# setwd(datafolder)
### calculate julian date and modis time series day and add to data frame
deer.opening$juliandate <- as.POSIXlt(deer.opening$Date)$yday+1
deer.opening$modisTS<-deer.opening$juliandate/16+1

### exclude deer with too short data period or deer which are not collard in spring
deersub <- subset(subset(deer, !(deer$id %in% c("2007_Smrt_2926","2007_Rosa_2926",
                                                "2010_Charlotte_1467","2005_Franta_945",
                                                "2008_Manuela_4947", "2008_Vincek_949"))))

deersub@data <- droplevels(deersub@data)
deer <- deersub
deer.all.dates.gat <- subset(deer, !(deer$id %in% c("2006_Faballa_2299", "2006_Radek_948", "2006_Supergirl_2045")))
wildanimals <- subset(deer, (deer$id %in% c("2006_Faballa_2299", "2006_Radek_948", "2006_Supergirl_2045")))
deer.all.dates.gat@data <- droplevels(deer.all.dates.gat@data)
wildanimals@data <- droplevels(wildanimals@data)
# save(wildanimals, file=str_c(datafolder, "/data_wildanimals_NDVI.RData"))

### check ndvi at gatter
```

```
ndvi<-stack(list.files(ndviorigfolder, full.names=TRUE))
gatterndvi<-extract(ndvi, gatter, fun=mean)
gatterndvi<-gatterndvi[complete.cases(gatterndvi),]
save(gatterndvi, file=str_c(datafolder, "/gatterndvi.RData"))
# load(str_c(datafolder, "/gatterndvi.RData"))

### remove gatter which has no valid NDVI Values
# gatter<-gatter[-8,]
# rownames(gatterndvi)<-gatter$OBJECTID
# # save(gatterndvi, file=str_c(datafolder, "/gatterndvi_extract.RData"))
# load(str_c(datafolder, "/gatterndvi_extract.RData"))
# ### start of season (sos)
# sos<-stack(list.files(sosfolder,pattern="^Dtl_StartSeason_Y.*_GrowingSeasonA\\.tif", full.names=TRUE))
# gattersos<-extract(sos, gatter, fun=median)
# colnames(gattersos)<-2001:2011
# save(gattersos, file=str_c(datafolder,"/gattersos_extract.RData")) ### objectname = gattersos
load(str_c(datafolder,"/gattersos_extract.RData"))
#interpolate NA Values
gattersos[gattersos>150]<-NA
gsa<-t(na.approx(t(gattersos), na.rm=T))
for(r in which(!complete.cases(gsa))){
    gsa[r, is.na(gsa[r,])]<-mean(gsa[r,], na.rm=T)
  }
gattersos<-gsa
gsa <- as.data.frame(gsa)
colnames(gsa) <- as.character(seq(2001,2011,1))
gsa.mon <- as.matrix(gsa)
###------------------------------------------------------------------------------
#####     END OF SCRIPT     ##########################################################
#####

  ###
  ### dates_winter_enclosure_opening.r
  ###

  library(foreign)
  library(rgdal)
  library(stringr)

  mainpath <- "D:/WORK/DATA/01_GPS/GPS_DATA_HIRSCH/project_reddeer_NDVI"
  data.encl <- paste0(mainpath, "/01_data/03_winter_enclosures")
  encl.shp <- readOGR(dsn=data.encl, layer="enclosures_BWCZ_UTM")
```

```
enc <- read.dbf(paste0(data.encl, "/JAHRESPR_NDVI.DBF"))
### change col names
colnames(enc) <- c("WINTERGATT", "year_1", "year_2","day_closed", "month_closed", "year_closed",
                   "day_open", "month_open", "year_open")
### rename names in enc$WINTERGATT
enc$WINTERGATT <- as.character(enc$WINTERGATT)
enc$OBJECTID <- NA
for(i in 1:nrow(enc)){
    if(enc$WINTERGATT[i] == "WG Ahornschachten"){enc$WINTERGATT[i] <- str_replace_all(
        enc$WINTERGATT[i], "WG Ahornschachten", "Ahornschachten")}
    if(enc$WINTERGATT[i] == "Ahornschachten"){enc$OBJECTID[i] <- 14}
    if(enc$WINTERGATT[i] == "WG Buchenau"){enc$WINTERGATT[i] <- str_replace_all(
        enc$WINTERGATT[i], "WG Buchenau", "Buchenau")}
    if(enc$WINTERGATT[i] == "Buchenau"){enc$OBJECTID[i] <- 15}
    if(enc$WINTERGATT[i] == "WG Neuhttenwiese"){enc$WINTERGATT[i] <- str_replace_all(
        enc$WINTERGATT[i], "WG Neuhttenwieseu", "Neuhuettenwiese")}
    if(enc$WINTERGATT[i] == "Neuhuettenwiese"){enc$OBJECTID[i] <- 16}
    if(enc$WINTERGATT[i] == "WG Riedlh"ng"){enc$WINTERGATT[i] <- str_replace_all(
        enc$WINTERGATT[i], "WG Riedlh"ng", "Riedlhaeng")}
if(enc$WINTERGATT[i] == "Riedlhaeng"){enc$OBJECTID[i] <- 17}
}
### Neuhuettenwiese and Riedlhaeng is not working, maybe due to unusual encoding,
### will change it manually
enc$WINTERGATT[13:20] <- "Neuhuettenwiese"
enc$OBJECTID[13:20] <- 16
enc$WINTERGATT[21:27] <- "Riedlaeng"
enc$OBJECTID[21:27] <- 17
### create POSIX for opening dates
enc$opening <- as.POSIXct(str_c(enc$year_open, "-", enc$month_open, "-", enc$day_open),
                          tz="UTC", format="%Y-%m-%d")
### create POSIX for opening dates
enc$closing <- as.POSIXct(str_c(enc$year_closed, "-", enc$month_closed, "-", enc$day_closed),
                          tz="UTC", format="%Y-%m-%d")
# save(enc, file=str_c(data.encl,"/enclosure_opening_dates.RData"))
###-------------------------------------------------------------------------------------
######     END OF SCRIPT     ##########################################################
######
######
### NDVI_MAX_slope.r
###
```

```
### Script to calculate the maximum increase in a (ndvi) time series
##########################################################################################
library("stringr")
library("raster")
years<-2001:2011
infolder <- "../01_data/02_NDVI/cropped_NDVI"
outfolder1 <- "../01_data/02_NDVI/max_slope_ndvi"
outfolder2 <- "../01_data/02_NDVI/slope_ndvi"
outfolder3 <- "../01_data/02_NDVI/max_slope_ndvi_filtered"
for(y in years){
    f<-list.files(infolder, pattern=str_c(y, ".*tif$"), full.names = T)
    ndvi<-stack(f)
    ### max slope
    #ndvi_slope_max<-calc(ndvi, fun=function(x) max.col(x[,-1]-x[,-ncol(x)]),
      filename=str_c(outfolder1,"/",str_replace(basename(f[1]),".tif", "_max_slope.tif" ) ),
    progress='text', format='GTiff', forcefun=T, overwrite=T)
    # filter for ndvi with less than delta ndvi value range
    delta<-200
    #following calculation might give warnings, this is due to the case that there are only Na
    ## values in a pixel, then range will give INF, -INF, this case is caught later on!
    ndvi_slope_max_filtered<-calc(ndvi, fun=function(x){
        mcol<-max.col(x[,-1]-x[,-ncol(x)])
        minmax<- t(apply(x, 1, function(y) range(y, na.rm=T)))
        mcol[minmax[,2]-minmax[,1]<delta | !is.finite(minmax[,2])]<-NA
        return(mcol)
    }, filename=str_c(outfolder3,"/",str_replace(basename(f[1]),".tif", "_max_slope_filtered.tif" )),
    progress='text', format='GTiff', forcefun=T, overwrite=T, progress=T)

    ### all slopes
    #ndvi_slope_all<-calc(ndvi, fun=function(x) x[,-1]-x[,-ncol(x)],
      filename=str_c(outfolder2,"/",str_replace(basename(f[1]),".tif", "_all_slopes.tif" ) ),
    progress='text', format='GTiff', forcefun=T, overwrite=T)
}
print("Done Computation of slope")
##------------------------------------------------------------------------------------
#####     END OF SCRIPT     ##########################################################
##########################################################################################


    ###
    ### extract_opening_dates.r
    ###
    ##########################################################################################
```

```
###     LIBRARIES                                                                            ###
################################################################################################
library("stringr")
library("raster")
library("rgdal")
library("sp")
library("rgeos")
library("adehabitatLT")
library("trip")
library("maptools")
setwd(mainfolder)

###     START     ##############################################################################
### subset
animals <- subset(deersub, deersub$month %in% c("Feb","Mar","Apr","May", "Jun", "Jul"))
animals@data <- droplevels(animals@data)
in.enc <- subset(animals, animals$id %in% names.animals.in.enclosure)
in.enc@data <- droplevels(in.enc@data)

### the data shows that animals have location points within and outside the enclosure on
### the same day. to define the date of the opening of winter enclosures we will check the
### sequence of 10 location points in a row that are not within the winter enclosures
reallygone <- list()
### subset over ids
for(j in 1:nlevels(animals$id)){
    sub2 <- subset(animals, animals$id == levels(animals$id)[j])
    sub2 <- sub2[order(sub2$DateTime),]
    for (i in 1:(dim(sub2)[1]-10)) {
        ### NA in column GATTERID means point is not within the polygon -> animal is not
        ### within the enclosure
        ### subset date when animals leaves enclosure
        if (sum(is.na(sub2$gatterid[i:(i+9)]))==10) {
            break
        }
    }
    ### extract Date and GATTERID: add to list
    DateTime <- as.character(sub2$DateTime[i])
    y <- factor(sub2$gatterid)
    GATTERID <- levels(y)[1]
    id <-  levels(animals$id)[j]
    xy <- cbind(DateTime,GATTERID,id)
    reallygone[[levels(animals$id)[j]]] <- xy
```

```
}
a <- as.data.frame(do.call(rbind, reallygone))
##-------------------------------------------------------------------------------------------------

    ### remove animals which are not in winter enclosures during winter
    a.not.i.gat <- subset(a, !(a$id %in% c("2006_Faballa_2299", "2006_Radek_948", "2006_Supergirl_2045")))
    deer1 <- droplevels(a.not.i.gat)
    deer.opening <- deer1
    save(deer.opening, file=str_c(datafolder,"/deer_date_enclosuers_open.RData" ))
    ##-------------------------------------------------------------------------------------------------
    ######     END OF SCRIPT     ##########################################################
```

Chapter 2 - Landscape configuration is a major determinant of home range size variation

```
###
### 01_settings_analysis_reddeer_homerange_landscape.r
###
###
### input data: one data.frame with all individuals. The object is called 'animals'
### and is stored as an .RData file in projectfolder/01_data/GPSdata
### important columns in input data are: coordinates (e.g. Easting and Northing), ID, month, week of year,
### biweek of year. Column names are specified in section "Settings for GPS data" in this script.
### Data was cleaned beforehand: all missing fixes (NAs, spatial & temporal false fixes) were removed
###
### important information for correct data handling: projection string and time zone specified in
### GPS collars need to be set in section "Settings for GPS data" in this script
###
### The data preparation script checks for > 5 locations per individual per time unit and creates an S4
### object for home range calculation (at least 5 locations are necessary to calculate a home range)
###
### Note: the input data frame must be named "animals"
###
### Environmental information used in this script: elevational data from ASTER
### and vegetational information from the study area
### (stored in separate folders)
###
### Additional software executed from this script: GRASS 6.4.1 and FRAGSTATS v3.3 - must be installed
###
#################################################################################################
#################################################################################################
#################################################################################################
###    PATHS                                                                                  ###
#################################################################################################
#################################################################################################
### for customizing script, the path ('mainfolder') needs to be adjusted and path 'inputdatafolder'
### needs to be set to the folder where the .RData file with the location points of animals to analyse
### is stored  (preferably the data should be stored in projectfolder/01_data/GPSdata)
mainfolder <- "D:/WORK/DATA/01_GPS/GPS_DATA_HIRSCH/project_reddeer_homerange_landscape"
### path to data folder
datafolder <- paste(mainfolder, "01_data", sep="/")
inputdatafolder <- paste(datafolder, "GPSdata", sep="/")    ### cleaned data set from original files
### path to scripts
codefolder <- paste(mainfolder, "02_code", sep="/")
sourcefolder <- paste(mainfolder, "02_code/source", sep="/")
```

```
### path to elevational information/data
asterfolder <- "D:/WORK/DATA/02_RS_GIS/GeodatenShare/ASTER/ASTERprocessing"
### path to rasters with information on vegetation. In this case seven vegetation classes for all single years
landusefolder <- "D:/WORK/DATA/02_RS_GIS/GeodatenShare/reclassification_VegetationMaps/04_output/VegetationMaps_cat7"
### fragstats folders (folder will be created automatically)
setwd(mainfolder);dir.create("fragstat_execution", showWarnings=FALSE)
fragexefolder<-paste(mainfolder, "fragstat_execution",sep="/")
setwd(fragexefolder);dir.create("frag_output", showWarnings=FALSE)
outputDummyfolder <- paste(fragexefolder, "frag_output", sep="/")
### folder to store workspace and intermediate steps (folder will be created automatically)
setwd(mainfolder) ; dir.create("03_RDatafiles",showWarnings = FALSE)
RDatafolder <- paste(mainfolder, "03_RDatafiles", sep="/")

#########################################################################################
###    LIBRARIES                                                                      ###
#########################################################################################
setwd(sourcefolder);source("libraries.r")### necessary libraries will be installed and loaded automatically
                                ### Information on versions of packages are stored in the script

#########################################################################################
###     SETTINGS FOR GPS DATA                                                         ###
#########################################################################################
### specify species (as.character)
species <- "reddeer"
### format Date and Time: depending on the GPS-collars the right time zone (tz) must be set
tz <- "GMT"
### projection string: depending on the collar settings (or data preprocessing)
workproj <- "+proj=utm +zone=33 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +towgs84=0,0,0"
### input data to analyse: input data is one data.frame with all GPS locations for all animals to analyse.
### Name of data.frame stored as .RData file is "animals"
### Within the data.frame the columns of the coordinates, Date and Time
### (as one column and formatted as POSIXct),
### id (in following order: collar number, deer name, year, separated with "_")
### and column which specifies the time unit
### (e.g. month name or week number) should be present
input.data.gps <- "RandomSampledLocations_hrlanduse_n38.RData"
### the column names of the coordinates and animal IDs need to be set (according to the data.frame)
coordcolumnnames <- c("Easting","Northing")
IDcolumnname <- "id"
### get timespan (years) of whole data set
setwd(inputdatafolder) ; load(input.data.gps)
timespan1 <- as.numeric(format(animals$DateTime, format="%Y"))
```

```
timespan2 <- as.data.frame(table(timespan1))
timespan <- timespan2[,1]
### percentclasses to calculate kernels
percentclasses <- c(50,70,90)
### set grid and extent for home range size calculation
myextent=20 ; mygrid=2000

##############################################################################################
###     SETTINGS FOR CALCULATION OF MEAN ELEVATION of GPS location of individual animals per timeunit     ###
##############################################################################################
elevation.filename <- "crop_aster_utm.tif"    ### filename of input raster with elevational information

##############################################################################################
###     SETTINGS FOR CALCULATION OF VEGETATION within a home range                         ###
##############################################################################################
### names of vegetation classes within the raster
vegclass.names <- c("coniferous","deciduous","regeneration","meadows","mixed","anthropogenic","other")
### pattern to load all rasters (the full name will be pasted with the respective year)
name.veg.rast <- "VegetationMap_cat7_"

##############################################################################################
###     SETTINGS FOR SPGRASS6 SESSION - CONVERTING RASTER TO ASCII FILE WITH AND WITHOUT HEADER    ###
##############################################################################################
### setting to load GRASS in R via spgrass6
memory.limit(4095)
mygrass <- "C:/Programme/GRASS-64"              ### path to GRASS GIS
mygisDbase <- "D:/WORK/DATA/02_RS_GIS/R_GRASS/GRASS_DATA_BASE"    ### path to GRASS GIS DataBase
mylocation <- "UTM_WGS84_33N_BayWa"             ### name of location
mymapset <- "Bayerwald"                         ### name of mapset
myres <- "5"                                    ### resolution
mynullvalue <- "999"                            ### null value to be set in ascii files

##############################################################################################
###     FRAGSTATS SESSION - CALCULATION OF LANDSCAPE METRICS                                ###
##############################################################################################

**********************************************************************************************
# ***   settings for calculation of landscape metrics                                        ***
**********************************************************************************************
### for batch file creation
root <- "batch"                         ### prefix for batchname
suffix <- ".fbt"                        ### file format
```

```
value.cellsize <- as.numeric(myres)          ### defined in settings SPGRASS6 session
value.background <- as.numeric(mynullvalue) ### defined in settings SPGRASS6 session
value.rows <- 555                            ### dummy value
value.columns <- 555                         ### dummy value
value.InputDataType <- "IDF_ASCII"           ### name of input data files
### for fragstats execution
batch.dummy <- "FragBatchDummy.fbt"          ### dummy to run batch script in Fragstats
myfragstat <- "C:/FragStats/Fragstats.exe"   ### path to fragstats programm

### Note: Parametrization.frg must be created manually in Fragstats (specifies input, output and indices) and
### needs to be stored in the folder "fragstats execution" within "mainfolder"
### settings in FragStats
### - path needs to be set to batch file.
### Folder = "fragstat_execution", name of batchfile = "FragBatchDummy.fbt"
### - name of output file = "Output_Batch_Fragstats" stored in subfolder "frag_output"
### - analysis type: Standard
### - class properties file: "classprop.fdc" in folder "fragstat_execution"
###   (specifies vegetation input classes, created manually)
### - Patch Neighbours: 8 Cell Rule
### - Output Statistic: Landscape Metrics - select indices of interest

### Fragstats Parametrization file
frag.parametrization <- str_c(fragexefolder,"/","Parametrization.frg", " "," "/c")
### output file name
frag.outputfile <- "Output_Batch_Fragstats.land"
#*****************************************************************************************************
#*****************************************************************************************************
#*****************************************************************************************************
### see also: http://www.umass.edu/landeco/research/fragstats/fragstats.html


#*****************************************************************************************************
#*********     RUN SCRIPTS     ***********************************************************************
#*****************************************************************************************************


#####################################################################################################
###    SETTINGS FOR MONTHLY TIMESCALE                                                             ###
#####################################################################################################
### create a folder for the timeunit
setwd(mainfolder) ; dir.create("04_timescale_monthly",showWarnings = FALSE)
timeunitfolder <- paste(mainfolder, "04_timescale_monthly", sep="/")
setwd(sourcefolder) ; source("additional_paths.r")     ### additional folders and paths will be created
timeunit.colname <- "Monat" ### column name for the time unit of interest as defined in the data (as.character)
```

```
stamp <- factor("monthly")  ### name of the time stamp of data (as factor) to analyse
temp.res <- 30              ### temporal resolution in days of the time unit

### data preparation
### this script will check for at least 5 relocations per individual per time unit and organize the data for
### kernel calculation.
setwd(codefolder) ; source("data_preparation_monthly_timescale.r")

### start analysing steps
###   - calculation of mean elevation of all GPS locations from individual animals per time unit
###   - calculation of home range size with kernel method / calculation of home range centre
###   - calculation of vegetation type within a home range
###   - rasterization of home ranges / conversions to ascii files
###   - calculation of landscape metrics with FragStats
start.time <- Sys.time()
gcinfo(verbose=FALSE)
setwd(codefolder) ; source("02_mainscript_running_analyses_homerange_landscape.r")
end.time <- Sys.time()
print(str_c("Start of script",":"," ", start.time)) ; print(str_c("End of script",":"," ", end.time))

### save workspace
setwd(RDatafolder) ;  save.image(str_c("workspace","_", species,"_", stamp, ".RData"))
### remove some objects from workspace
rm(pixsub,tier.count.proz.7,veg.count.7,hr.count_all,hr.centre.bb,hr.centre,hr.centre.kernel); gc()

### created output files in folder 'path/projectfolder/04_timescale_monthly/output_files'
### all file names in the following lines (after the -> ) refer to species: 'reddeer' and timescle: 'monthly'
### 1) mean elevation in home range -> 'elevation_monthly_timescale_reddeer.txt'
### 2) home range size - kernel method -> 'reddeer_hrsizes_monthly_kernel_method.txt
### 3) home range centre - kernel method -> 'reddeer_hrcentre_monthly_kernel_method.txt'
### 4) vegetation within home range - for kernel  -> 'reddeer_monthly_veg_in_hr.txt'
### 5) landscape indices - for kernel -> 'landscape_indices_reddeer_monthly.txt'
###
### Note: script contains a subset on columns (with column names) for information on sex, age, ...
### -> possible error for other input data (row 41)
### Note: additional data (climate variables) will be added. Paths are defined within the script
setwd(codefolder) ; source("03_stack_output_files_plus_other_data.r")


##############################################################################################
###    SETTINGS FOR BIWEEKLY TIMESCALE                                                     ###
##############################################################################################
```

```
### for detailed comments see section "SETTINGS FOR MONTHLY TIMESCALE"
### create folder for timeunit
setwd(mainfolder) ; dir.create("05_timescale_biweekly",showWarnings = FALSE)
timeunitfolder <- paste(mainfolder, "05_timescale_biweekly", sep="/")
setwd(sourcefolder) ; source("additional_paths.r")
prefix <- "biweek"                    ### for list of data sorted by timeunit
timeunit.colname <- "biweekofyear"    ### as defined in the data
stamp <- factor("biweekly")
temp.res <- 14
## data preparation
setwd(codefolder) ; source("data_preparation_biweekly_timescale.r")
## start analysing steps
start.time <- Sys.time()
setwd(codefolder) ; source("02_mainscript_running_analyses_homerange_landscape.r")
end.time <- Sys.time()
print(str_c("Start of script",":"," ", start.time)) ; print(str_c("End of script",":"," ", end.time))
### save workspace
setwd(RDatafolder) ;  save.image(str_c("workspace","_", species,"_", stamp, ".RData"))
### stack output data
setwd(codefolder) ; source("04_stack_output_files_plus_other_data.r")
### remove some objects from workspace
rm(animal, allrast, data.bb, data.kernel, hrlist_all)

#########################################################################################
###                                                                                   ###
###    SETTINGS FOR WEEKLY TIMESCALE                                                  ###
###                                                                                   ###
#########################################################################################
### for detailed comments see section "SETTINGS FOR MONTHLY TIMESCALE"
### create folder for timeunit
setwd(mainfolder) ; dir.create("06_timescale_weekly",showWarnings = FALSE)
timeunitfolder <- paste(mainfolder, "06_timescale_weekly", sep="/")
setwd(sourcefolder) ; source("additional_paths.r")
prefix <- "week"                      ### for list of data sorted by time unit
timeunit.colname <- "weekofyear"      ### as defined in the data
stamp <- factor("weekly")
temp.res <- 7
### data preparation
setwd(codefolder) ; source("data_preparation_weekly_timescale.r")
## start analysing steps
start.time <- Sys.time()
setwd(codefolder) ; source("02_mainscript_running_analyses_homerange_landscape.r")
end.time <- Sys.time()
print(str_c("Start of script",":"," ", start.time)) ; print(str_c("End of script",":"," ", end.time))
```

```
### save workspace
setwd(RDatafolder) ;  save.image(str_c("workspace","_", species,"_", stamp, ".RData"))
### stack output data
setwd(codefolder) ; source("04_stack_output_files_plus_other_data.r")

#########################################################################################################
### SETTINGS FOR INDEX VALIDATION                                                                     ###
#########################################################################################################
### create folder and paths
setwd(mainfolder) ; dir.create("validate_index", showWarnings=FALSE)
valindexfolder <- str_c(mainfolder,"/","validate_index")
setwd(valindexfolder) ; dir.create("cropped_buffers", showWarnings=FALSE)
dir.create("ascii_files", showWarnings=FALSE)
dir.create("ascii_with_header", showWarnings=FALSE)
dir.create("raster", showWarnings=FALSE);dir.create("fragstats", showWarnings=FALSE)
dir.create("output", showWarnings=FALSE)
croppedBuffs <- str_c(valindexfolder, "/","cropped_buffers")
ascbuffs <- str_c(valindexfolder, "/", "ascii_files")
headascbuffs <- str_c(valindexfolder, "/", "ascii_with_header")
rasterbuffs <- str_c(valindexfolder, "/", "raster")
fragbufffolder <- str_c(valindexfolder, "/", "fragstats")
outputvalindex <- str_c(valindexfolder, "/", "output")
### buffer sizes to draw around  home range centres to validate index
buffersizes <- seq(500,7000,500)
### define data
### the buffers around the hr centres will be drawn from the 90% kernel hr centre points,
### to keep the calculation time reasonable. We can further define from which time scale we
### take the hr centres. Here I choose the monthly time scale. We need to specify
### the path to the outputfolder of the specific time scale we choose. Possible folder are:
### 04_timescale_monthly/output_files
### 05_timescale_biweekly/output_files
### 06_timescale_weekly/output_files
outputfolder_val <- str_c(mainfolder, "/", "04_timescale_monthly/output_files")
setwd(outputfolder_val)
stamp <- factor("monthly")
### get data
a <- read.table(str_c(species, "_hrcentre_", stamp, "_kernel_method.txt"), sep="\t",header=TRUE)
hr.centre_all<- rbind(a,b)
hr.centre_df_k90 <- hr.centre_all[which(hr.centre_all$size =="90" & hr.centre_all$method == "kernel"),]
hr.centre_df <- hr.centre_df_k90
### run scripts
Sys.time()
```

```
setwd(codefolder) ; source("03_landscape_index_validation.r")
Sys.time()
### save wprkspace
setwd(RDatafolder) ;  save.image(str_c("workspace_index_validation_", species,"_", stamp, ".RData"))

############################################################################################################
#####       END OF SCRIPT                                                                              #####
############################################################################################################


###
### 02_mainscript_running_analyses_homerange_landscape.r
###
### script to execute single analysing steps
###   - calculation of mean elevation of all GPS locations from individual animals per time unit
###   - calculation of home range size with kernel method / calculation of home range centre
###   - calculation of vegetation type within a home range
###   - rasterization of home ranges / conversions to ascii files
###   - calculation of landscape metrics with FragStats
###
############################################################################################################

############################################################################################################
###    CALCULATION OF MEAN ELEVATION of GPS location of individual animals per time unit                 ###
############################################################################################################
### to evaluate the mean elevation of home ranges of single animals, all location points are drawn on an
### elevational map and the values are extracted behind these locations.
### Then the mean per time unit is calculated.

#***********************************************************************************************************
# ***   settings for elevation calculation                                                              ***
#***********************************************************************************************************
### output file name to store results (in folder 'outputfolder')
output.elevation <- str_c("elevation","_", stamp,"_", "timescale", "_", species, ".txt")
### Note the name of the input data frame (with telemetry data) is "animals"
#***********************************************************************************************************
#***********************************************************************************************************

### get animal data
setwd(inputdatafolder) ; load(input.data.gps)
### get elevation map - name of layer is defined in "01_settings_xxx.r"
setwd(asterfolder) ; aster <- raster(paste(asterfolder, elevation.filename, sep="/"))
```

```
### define timeunit as factor (colname specification is set under script "01_settings_xxx.r")
timeunit <- factor(animals[[timeunit.colname]])
### run calculation and store results
print("calculating mean elevation")
setwd(sourcefolder); source("calculate_elevation.r")
### name of output data frame = elevation
elevation <- as.data.frame(do.call(rbind, elevlist))
colnames(elevation) <- c("elev", "id", "timeunit","stamp")
setwd(outputfolder); write.table(elevation, output.elevation, sep="\t", row.names=FALSE)
###------------------------------------------------------------------------------------------
rm(animals, mon1, datasub, mon2, aster, elevation, elevdf, elevlist) ; gc()

#############################################################################################
###                                                                                       ###
###     KERNEL CALCULATION                                                                ###
###                                                                                       ###
#############################################################################################
### calculate the home range size with the kernel method
### the percent classes are defined in "01_settings_xxx.r"

#*******************************************************************************************
# *** settings for kernel calculation                                                   ***
#*******************************************************************************************
### path to store shape files (string) is defined in "01_settings_xxx.r",
### source code "additional paths" respectively
path = singleshpkernel
### name of output file of the calculated kernel areas
outputfile.hrsize.kernel.method <- paste(species, "hrsizes", stamp, "kernel_method.txt", sep="_")
### name of outputfile of the calculated home range centres for kernel method
outputfile.hrcentre.kernel <- paste(species, "hrcentre", stamp, "kernel_method.txt", sep="_")
### id names for the different kernelsizes
kernelid <- vector()
for(i in 1:length(percentclasses))kernelid[i] <- str_c("k_","hr",percentclasses[i])
method.id <- kernelid
### name of the .RData file where all kernelareas (as polygons) over all 'percentclasses' are stored
hrlist.name.kernel <- paste(species, "hrlist", stamp,"all_kernels.RData", sep="_")
### general method name
method.name <- "kernel"
#*******************************************************************************************

### inputdata for kernel calculation
### (created with script "data_preparation_xxx_timescale.r, where xxx = timeunit)
### class of inputdata = list sorted by timeunit (e.g. months).
### name of list = animals, class = SpatialPointsDataFrame
```

```
setwd(RDatafolder) ; load(data.kernel)
### calculate home range sizes with kernel method and save output file
print("calculating home range sizes with kernel method") ; print("Start of Calculation:") ; print(Sys.time())
setwd(sourcefolder) ; source("animalkernelUD.r")
kernelarea<-list()
for (i in as.character(percentclasses))  kernelarea[[i]] <- data.frame()
for (animal in 1:length(animals)){
    tmp <- animalkernelUD(animals[[animal]][,IDcolumnname], timeunit = names(animals)[animal], stamp,
      path,percentclasses)
      for (i in as.character(percentclasses)) kernelarea[[i]]<-rbind(kernelarea[[i]],tmp[[i]])
  }
setwd(outputfolder)
write.table(as.data.frame(do.call(rbind, kernelarea)), file=outputfile.hrsize.kernel.method,
  row.names=FALSE, sep="\t")
### create a list of all polygons and save them
setwd(sourcefolder) ; source ("hrlist.r")
setwd(RDatafolder) ; hrlist_all_kernel <- hrlist; save(hrlist_all_kernel, file=hrlist.name.kernel)
hrlist <- hrlist_all_kernel
###-----------------------------------------------------------------------------------------------
### calculation of home range centre
setwd(sourcefolder); source("hrcentre.r")
setwd(outputfolder); write.table(as.data.frame(do.call(rbind, hr.centre)), file=outputfile.hrcentre.kernel,
                                 row.names=FALSE, sep="\t")
hr.centre.kernel <- hr.centre
###-----------------------------------------------------------------------------------------------
print("End of Calculation:") ; print(Sys.time())
rm(hrlist, animals, animalkernelUD, hrlist_one, kernelarea) ; gc()

#########################################################################################
###    CALCULATION OF VEGETATION within a home range                                  ###
#########################################################################################

#*****************************************************************************
# ***   settings for vegetation calculation                                  ***
#*****************************************************************************
### name of output file
veg.in.hr.output <- paste(species, stamp, "veg_in_hr.txt", sep="_")
### name to store calculation 'pixel of vegetation within a homerange'
output.pixelcount <- paste(species, stamp, "pixlist_all.RData",sep="_")
#*****************************************************************************
#*****************************************************************************
```

```
### setwd(RDatafolder) ; load(hrlist.name.kernel)  ### run this line if data is not in memory
hrlist_all <- hrlist_all_kernel
print("calculating vegetation within home range"); print("Start of Calculation:") ; print(Sys.time())
setwd(sourcefolder) ; source("veg_in_hr.r")
### data table
setwd(outputfolder) ; write.table(veg.df, veg.in.hr.output,row.names=FALSE,sep="\t")
##-----------------------------------------------------------------------------
print("End of Calculation:") ; print(Sys.time())
rm(pixlist_all, pixlistcat7, pixlist1, mylist, hrlist, landmap, veg.df); gc()

#############################################################################
###                                                                       ###
###     RASTERISATION OF HOME RANGES                                      ###
###                                                                       ###
#############################################################################

### run following two lines if data is not in memory
### setwd(RDatafolder) ; load(hrlist.name.kernel)
### hrlist_all <- hrlist_all_kernel
##-----------------------------------------------------------------------------
print("rasterization of home ranges") ; print("Start of Calculation:") ; print(Sys.time())
setwd(sourcefolder) ; source("homerange_rasterisation.r")
##-----------------------------------------------------------------------------
print("End of Calculation:") ; print(Sys.time())
ls1 <- ls(pattern="hrlist") ; rm(list=ls1) ; rm(r4, b, yearlist, r, landmap,a ) ; gc()

#############################################################################
###                                                                       ###
###     SPGRASS6 SESSION - CONVERTING RASTER TO ASCII FILE WITH AND WITHOUT HEADER
###                                                                       ###
#############################################################################

#*****************************************************************************
# ***   settings for spgrass6 session                                       ***
#*****************************************************************************
### paths to folders
targetfolder <- allrast
output.asc.without.header <- ascfolders
output.asc.with.header <- headascfolderall
#*****************************************************************************
#*****************************************************************************

print("converting raster to ascii") ; print("Start of Calculation:") ; print(Sys.time())
setwd(sourcefolder) ; source("SPGRASS6_session_convert_raster_to_ascii_with_and_without_header.r")
print("End of Calculation:") ; print(Sys.time())
```

```
##########################################################################################################################
###     FRAGSTATS SESSION - CALCULATION OF LANDSCAPE METRICS                                                           ###
##########################################################################################################################

  ************************************************************************************************************************
  # ***   settings for calculation of landscape metrics                                                                  ***
  ************************************************************************************************************************
  ### stamp    ### defined in settings
  ### kernelid ### defined in kernel calculation
  nameid <- kernelid
  #method.name <- c(rep("brownian",3),rep("kernel",3))
  method.names <- rep("kernel",3)
  percentclass2 <- rep(percentclasses,2)
  batchfilenames <- vector()
  ### for batch creation:
  for(i in 1:length(nameid)) batchfilenames[[i]] <- str_c(root,"_", nameid[i],"_",stamp,suffix)
  outputfile.fragstats <-  str_c("landscape_indices","_", species,"_",stamp, ".txt")
  targetbatch <- headascfolderall
  inputasc <- ascfolders
  fragbatchfolder <- fragfolderall
  ************************************************************************************************************************
  ************************************************************************************************************************

  print("calculating landscape metrics") ; print("Start of Calculation:") ; print(Sys.time())
  setwd(sourcefolder) ; source("FragStats_session.r")
  ### name of output data frame = frag.all
  ### reshaping output - adding additional columns
  split4 <- as.data.frame(str_split_fixed(frag.all$filename, pattern=str_c(as.character(stamp),"_"),2))
  split5 <- as.data.frame(str_split_fixed(split4[,2], pattern="_", 4))
  frag.all$timeunit <- split5[,4]
  frag.all$id <- str_c(split5[,1],"_", split5[,2],"_", split5[,3])
  split6 <- as.data.frame(str_split_fixed(split4[,1], pattern="_",3))
  frag.all$method <- split6[,1]
  frag.all$method <- recode(frag.all$method, "brownian" <- "bb", "kernel" <- "k",  otherwise="copy")
  split7 <- as.data.frame(str_split_fixed(split6[,2],pattern="hr",2))
  frag.all$size <- split7[,2]
  ### write output file
  setwd(outputfolder) ; write.table(frag.all, file= outputfile.fragstats, sep="\t")
  ###-----------------------------------------------------------------------------------------------------------------
  print("End of Calculation:") ; print(Sys.time())
  rm(all.land, frag.all, myrast) ; gc()
```

```
############################################################################################################
#####        END OF SCRIPT                                                                             #####
############################################################################################################


###
### 03_landscape_index_validation.r
###
### index validation
###
### taking the home range centres of der 90% kernelsfor calculation of
### defined areas (buffers) to analyse the size dependency of the landscape index.
###
############################################################################################################

############################################################################################################
###    VALIDATE LANDSCAPE METRICS - CREATING BUFFERS                                                     ###
############################################################################################################

#***********************************************************************************************************
#***   settings for validation of landscape metrics                                                     ***
#***********************************************************************************************************
targetfolder <- croppedBuffs
output.asc.without.header <- ascbuffs
output.asc.with.header <- headascbuffs
#***********************************************************************************************************
#***********************************************************************************************************
print("creating buffers around home range centres and rasterization of buffers")
setwd(sourcefolder) ; source("create_buffers_around_hrcentres.r")
rm(r4, b, buffers, pts) ; gc()

targetfolder <- rasterbuffs
print("converting raster to ascii")
setwd(sourcefolder) ; source("SPGRASS6_session_convert_raster_to_ascii_with_and_without_header.r")
rm(all.buffers, hrcentre.splist, yearlist, allpts, hr.centre_all, landmap, r, hrcentresub, a) ; gc()

#### running fragstats
batchfilenames <- str_c(root,"_", "validate_index","_",stamp,suffix)
outputfile.fragstats <-  str_c("validate_index","_", species,"_",stamp, ".txt")
targetbatch <- headascbuffs
inputasc <- ascbuffs
fragbatchfolder <- fragbufffolder
```

```
setwd(sourcefolder) ; source("FragStats_session.r")

### name of df = frag.all
### rehsping output - adding additional columns
split4 <- as.data.frame(str_split_fixed(frag.all$filename, pattern="_",7))
frag.all$timeunit <- split4[,3]
frag.all$stamp <- stamp
frag.all$id <- paste(split4[,4],split4[,5], split4[,6],sep="_")
frag.all$method  <- str_sub(split4[,7], end=-3L)
frag.all$size <-  str_sub(split4[,7],start=-2L)
frag.all$buffersize <- split4[,2]
setwd(outputvalindex) ; write.table(frag.all, file= outputfile.fragstats, sep="\t")

#########################################################################################
#####                                                                               #####
#####       END OF SCRIPT                                                           #####
#####                                                                               #####
#########################################################################################


###
### 04_stack_output_files_plus_other_data.r
###
### stack created output data files
###
### additional information about temperature, rain and daylight hours will be added at the end of the script
### here the paths need to be adjusted or lines need to be commented out
#########################################################################################

setwd(outputfolder)
### stack data kernel method
### elevation
elev <- read.table(str_c("elevation_", stamp, "_timescale_", species, ".txt"), sep="\t", header=TRUE)
### home range sizes - kernel method
hrsize.kernel <- read.table(str_c(species, "_hrsizes_", stamp, "_kernel_method.txt"), sep="\t", header=TRUE)
# colnames(hrsize.kernel) <- c("id","area","timestamp","timeunit","size","method")
hrsize.kernel$timeunit <- ifelse(hrsize.kernel$timestamp != "monthly", gsub("[^0-9]","", hrsize.kernel$timeunit),
                                 as.character(hrsize.kernel$timeunit))

### merge
data1.kernel <- merge(elev, hrsize.kernel, by=c("id", "timeunit"), sort=FALSE)
### home range centres - kernel method
hrcentre.kernel <- read.table(str_c(species, "_hrcentre_", stamp, "_kernel_method.txt"), sep="\t", header=TRUE)
hrcentre.kernel$timeunit <- ifelse(hrcentre.kernel$stamp != "monthly",gsub("[^0-9]","", hrcentre.kernel$timeunit),
                                   as.character(hrcentre.kernel$timeunit))
```

```
### merge
data2.kernel <- merge(data1.kernel, hrcentre.kernel, by=c("id", "timeunit","size","method", "stamp"), sort=FALSE)
### vegetation
veg <- read.table(str_c(species, "_", stamp, "_veg_in_hr.txt"), sep="\t", header=TRUE)
veg$stamp <- stamp
veg$timeunit <- ifelse(veg$stamp != "monthly",gsub("[^0-9]","", veg$timeunit), as.character(veg$timeunit))
veg$stamp <- NULL
### merge
data3.kernel <- merge(data2.kernel, veg, by=c("id", "timeunit","size","method"), sort=FALSE)
### landscape indices
land <- read.table(str_c("landscape_indices_",species, "_", stamp, ".txt"), sep="\t", header=TRUE)
land$stamp <- stamp
land$timeunit <- ifelse(land$stamp != "monthly",gsub("[^0-9]","", land$timeunit), as.character(land$timeunit))
land$stamp <- NULL
### merge
data4.kernel <- merge(data3.kernel, land, by=c("id", "timeunit","size","method"), sort=FALSE)
### get sex,age,...
setwd(inputdatafolder)
load(input.data.gps)
### get information of interest
gags <- subset(animals, animals$id == levels(animals$id)[1])
mysub <- gags[1,]
for(i in 2:nlevels(animals$id)){
    gags <- subset(animals, animals$id == levels(animals$id)[i])
    mysub <- rbind(mysub, gags[1,])
}
mysubsub <- mysub[, c("name", "year","sender", "Ohrmarke", "Geschlecht", "Altersklasse", "Alter", "id", "IDO")]
### merge
data5.kernel <- merge(data4.kernel, mysubsub,by.x=c("id", "year"), by.y=c("id", "year"))
### temporal resolution in days
data5.kernel$durationDays <- temp.res
### write merged data file to folder
setwd(outputfolder)
write.table(data5.kernel, file=str_c("stacked_outputfiles_", stamp, "_", species,"_kernel.txt"), sep="\t")
##########################################################################################################
### stack additional data
sunfolder <- "D:/WORK/DATA/03_Daten/daylight hours/output"
climatefolder <- "D:/WORK/DATA/03_Daten/Wetterdaten"
setwd(sunfolder)
sun <- read.table(str_c("daylighthours_", stamp, ".txt"), sep="\t", header=TRUE)
data6.kernel <- merge(data5.kernel, sun, by.x=c("timeunit", "year"), by.y=c("timeunit", "year"), sort=FALSE)
setwd(climatefolder)
```

```
climate <- read.table(str_c("Tmean_dev_Rmean_dev_",stamp, ".txt"), sep="\t", header=TRUE)
data7.kernel <- merge(data6.kernel, climate, by.x=c("year", "timeunit"), by.y=c("year", "timeunit"), sort=FALSE )
setwd(outputfolder)
write.table(data7.kernel, file=str_c("stacked_data_", stamp, "_", species,"_kernel.txt"), sep="\t")
######################################################################################################
######      END OF SCRIPT                                                                       ######
######################################################################################################


######################################################################################################
######      Data preparation files                                                              ######
######################################################################################################

######################################################################################################
###     DATA PREPARATION  -  MONTHLY TIMESCALE                                                     ###
######################################################################################################

#*****************************************************************************************************
# ***   settings for data preparation                                                             ***
#*****************************************************************************************************
data.kernel <- paste(species, "data", stamp, "stamp_kernel.RData", sep="_")
### file name to store prepared data for kernel analyses
#*****************************************************************************************************
#*****************************************************************************************************
### get data
setwd(inputdatafolder) ; load(input.data.gps)
###-----------------------------------------------------------------------------------------------------
### format Date and Time  NOTE: depending on the GPS-collars the right time zone (tz) must be set!
### (defined in settings file)
animals$DateTime <- as.POSIXct(strptime(paste(animals$DateTime), "%Y-%m-%d %H:%M:%S"), tz=tz)
###-----------------------------------------------------------------------------------------------------
### add month to data frame and create new ID
animals$month <- format(animals$DateTime, format = "%b")
###-----------------------------------------------------------------------------------------------------
### due to the system settings of my computer, I get a german spelling of the months.
### I need to convert them  to english spelling.
### If the computer settings are set to english the following lines can be skipped
animals$month <- replace(animals$month, animals$month == "Mrz", "Mar")
animals$month <- replace(animals$month, animals$month == "Mai", "May")
animals$month <- replace(animals$month, animals$month == "Okt", "Oct")
animals$month <- replace(animals$month, animals$month == "Dez", "Dec")
animals$month <- factor(animals$month)
```

```
animals$id_month <- factor(paste(animals[[IDcolumnname]], animals$month, sep="_"))
##------------------------------------------------------------------------------------------------
### at least 5 relocations are needed to calculate home range size, so we first need to check the data and
### skip data were we do not have > 5 relocations per individual per timeunit
countidmonth <-aggregate(rep(1,length(animals$id_month)), by=list(animals$id_month),FUN=sum)
countidmonth<-countidmonth[countidmonth$x>5,1]
animalsub <-animals[animals$id_month %in% countidmonth,]
animalsub <- droplevels(animalsub)
animals.data <-list()
for(i in unique(animalsub$month)) animals.data[[i]]<-droplevels(animalsub[animalsub$month == i,])
animals <- animals.data
### project monthly subsets for kernel calculation NOTE: the right projection string need to be set
### (defined in settings file)
### NOTE: the right column names of the coordinates need to be set (defined in settings file)
animals<-lapply(animals,function(x){ coordinates(x) <-coordcolumnames
proj4string(x) <- CRS(workproj); return(x)} )
setwd(RDatafolder) ; save(animals, file=data.kernel)
##------------------------------------------------------------------------------------------------

##################################################################################################
###     DATA PREPARATION    -     BIWEEKLY TIMESCALE                                           ###
##################################################################################################
# ***   settings for data preparation                                                          ***
data.kernel <- paste(species, "data", stamp, "stamp_kernel.RData", sep="_")
### file name to store prepared data for kernel analyses
### get data
setwd(inputdatafolder) ; load(input.data.gps)
###-----------------------------------------------------------------------------------------------
### format Date and Time  NOTE: depending on the GPS-collars the right time zone (tz) must be set!
### (defined in settings file)
animals$DateTime <- as.POSIXct(strptime(paste(animals$DateTime), "%Y-%m-%d %H:%M:%S"), tz=tz)
###-----------------------------------------------------------------------------------------------
### calculate week of year (ISOweek)
setwd(sourcefolder) ; source("ISOweek.r")
animals$Date <- as.Date(animals$DateTime, "%Y-%m-%d", tz="GMT")
animlas$weekofyear<- (getweek(animals$Date))$ISOWeek
###-----------------------------------------------------------------------------------------------
### create biweekly stamp
```

```
animals$biweekofyear <- animals$weekofyear + animals$weekofyear %% 2
animals$biweekofyear <- factor(animals$biweekofyear)
###------------------------------------------------------------------------------
### at least 5 relocations are needed to calculate home range size,
### so we first need to check the data and skip data
### were we do not have > 5 relocations per individual per timeunit
animals[[IDcolumnname]] <- factor(animals[[IDcolumnname]])
animals$id_biweek <- factor(paste(animals[[IDcolumnname]], animals$biweekofyear, sep="_"))
###------------------------------------------------------------------------------
### data with biweekly stamp for kernel calculation (all relocations > 5)
countidbiweek <-aggregate(rep(1,length(animals$id_biweek)), by=list(animals$id_biweek),FUN=sum)
countidbiweek<-countidbiweek[countidbiweek$x>5,1]
animalsub <-animals[animals$id_biweek %in% countidbiweek,]
animalsub <- droplevels(animalsub)
animals.data<-list()
for(i in unique(animalsub$biweekofyear)){
    animals.data[[paste(prefix,i, sep="")]]<-droplevels(animalsub[animalsub$biweekofyear == i,])
    animals <- animals.data
}
### project biweekly subsets for kernel calculation (needs to be spatial)
### NOTE: the right column names of the coordinates need to be set
animals<-lapply(animals,function(x){ coordinates(x) <- coordcolumnnames
                                     proj4string(x) <- CRS(workproj); return(x)} )
setwd(RDatafolder)
save(animals, file=data.kernel)
###------------------------------------------------------------------------------

##############################################################################
###     DATA PREPARATION  -    WEEKLY TIMESCALE                            ###
##############################################################################
# ***   settings for data preparation                                      ***
##############################################################################
data.kernel <- paste(species, "data", stamp, "stamp_kernel.RData", sep="_")
### file name to store prepared data for kernel analyses
##############################################################################
##############################################################################
### get data
setwd(inputdatafolder) ; load(input.data.gps)
###------------------------------------------------------------------------------
### format Date and Time  NOTE: depending on the GPS-collars the right time zone (tz) must be set!
### (defined in settings file)
```

```
animals$DateTime <- as.POSIXct(strptime(paste(animals$DateTime), "%Y-%m-%d %H:%M:%S"), tz=tz)
###-------------------------------------------------------------------------------------------------
### calculate week of year (ISOweek)
setwd(sourcefolder) ; source("ISOweek.r")
animals$Date <- as.Date(animals$DateTime, "%Y-%m-%d", tz="GMT")
animals$weekofyear<- (getweek(animals$Date))$ISOWeek
###-------------------------------------------------------------------------------------------------
### create weekly stamp
animals$weekofyear <- factor(animals$weekofyear)
###-------------------------------------------------------------------------------------------------
### at least 5 relocations are needed to calculate home range size,
### so we first need to check the data and skip data
### were we do not have > 5 relocations per individual per timeunitlocations per individual per timeunit
### create ID with timestamp
animals[[IDcolumnname]] <- factor(animals[[IDcolumnname]])
animals$id_week <- factor(paste(animals[[IDcolumnname]], animals$weekofyear, sep="_"))
###-------------------------------------------------------------------------------------------------
### data with weekly stamp for kernel  calculation (all relocations > 5)
countidweek <-aggregate(rep(1,length(animals$id_week)), by=list(animals$id_week),FUN=sum)
countidweek<-countidweek[countidweek$x>5,1]
animalsub <-animals[animals$id_week %in% countidweek,]
animalsub <- droplevels(animalsub)
animals.data<-list()
for(i in unique(animalsub$weekofyear)){
} animals.data[[paste(prefix,i, sep="")]]<-droplevels(animalsub[animalsub$weekofyear == i,])
animals <- animals.data
### project monthly subsets for kernel calculation (needs to be spatial)
animals<-lapply(animals,function(x){ coordinates(x) <- coordcolumnnames
 proj4string(x) <- CRS(workproj); return(x)} )
setwd(RDatafolder)
save(animals, file=data.kernel)
###-------------------------------------------------------------------------------------------------

##############################################################################################
######      SOURCEFILES                                                                 ######
##############################################################################################
##############################################################################################

###
```

```
### additional_paths.r
###
### additional paths are needed - folders will be created automatically
### mainfolder, sourcefolder and inputdatafolder are defined in the settings file
###
##############################################################################################################

    ### create folders
    setwd(timeunitfolder);dir.create("kernel",showWarnings = FALSE)
    dir.create("output_files",showWarnings = FALSE)
    kernelfolder <- paste(timeunitfolder, "kernel", sep="/")
    outputfolder <- paste(timeunitfolder, "output_files", sep="/")
    ###-----------------------------------------------------------------------------------------------
    ### create folders for shapefiles from kernel method
    setwd(kernelfolder);dir.create("shapes",showWarnings = FALSE)
    shpfolderkernel<- paste(kernelfolder, "shapes", sep="/")
    setwd(shpfolderkernel)
    for(i in 1:length(percentclasses)) dir.create(paste("kernel", percentclasses[i], sep=""),showWarnings = FALSE)
    singleshpkernel <- vector()
    for(i in 1:length(percentclasses)) singleshpkernel[i] <- str_c(shpfolderkernel,
                                                                   "/", "kernel", percentclasses[[i]])

    ###-----------------------------------------------------------------------------------------------
    ### create folders for home range rasterisation, conversion to ascii files (with and without header),
    ## cropped raster and fragstats output
    ### for both methods
    # method.id <- c("bb", "k")
    method.id <- "k"
    setwd(kernelfolder) ; dir.create("landscape", showWarnings=FALSE)
    kernelland <- paste(kernelfolder, "landscape", sep="/")
    setwd(kernelland)
    for(i in 1:length(percentclasses)) dir.create(paste(method.id[[2]], percentclasses[i], sep=""),showWarnings = FALSE)
    kernellandsub <- vector()
    for(i in 1:length(percentclasses)) kernellandsub[i] <- str_c(kernelland, "/", method.id[[2]], percentclasses[[i]])
    setwd(bbfolder) ; dir.create("landscape", showWarnings=FALSE)
    bbland <- paste(bbfolder, "landscape", sep="/")
    setwd(bbland)
    for(i in 1:length(percentclasses)) dir.create(paste(method.id[[1]], percentclasses[i], sep=""),showWarnings = FALSE)
    bblandsub <- vector()
    for(i in 1:length(percentclasses)) bblandsub[i] <- str_c(bbland, "/", method.id[[1]], percentclasses[[i]])
    bbandk <- c(bblandsub, kernellandsub)
    for(i in 1:length(bbandk)){
        setwd(bbandk[[i]])
```

```
    dir.create("ascii_files", showWarnings=FALSE)
    dir.create("ascii_with_header", showWarnings=FALSE)
    dir.create("cropped_raster", showWarnings=FALSE)
    dir.create("fragstats", showWarnings=FALSE)
    dir.create("raster", showWarnings=FALSE)
}
allcropped <- vector() ; for(i in 1:length(bbandk)) allcropped[i] <- str_c(bbandk[[i]],"/", "cropped_raster")
allrast <- vector() ; for(i in 1:length(bbandk)) allrast[i] <- str_c(bbandk[[i]],"/", "raster")
headascfolderall <- vector() ; for(i in 1:length(bbandk)) headascfolderall[i] <- str_c(bbandk[[i]],"/", "ascii_with_header")
ascfolders <- vector() ; for(i in 1:length(bbandk)) ascfolders[i] <- str_c(bbandk[[i]],"/", "ascii_files")
fragfolderall <- vector() ; for(i in 1:length(bbandk)) fragfolderall[i] <- str_c(bbandk[[i]],"/", "fragstats")
##------------------------------------------------------------------------------------------------------------
#####     END OF SCRIPT     ##########################################################################
```

```
###
### animalkernelUD.r
###
### function to calculate kernel area of home ranges
### parameters to be set:
###   - animalIDs = as.list sorted by time unit (e.g. month)
###   - timeunit = period of time for which the kernel is calculated (e.g. "May") as. character
###   - stamp = timestamp as character (e.g. "monthly")
###   - percentclasses = the kernel size to be calculated (e.g 90% kernel)
###   - path = path to folder(s) where to store the shape files of the kernels
###
##############################################################################################################

animalkernelUD <- function(animalIDs, timeunit, stamp, path, percentclasses, grid=mygrid, extent=myextent,
                           h="href", kern="bivnorm"){
    hr <- kernelUD(animalIDs, grid=grid, extent=extent, h=h, kern=kern)
    kernelindex<-as.character(percentclasses)
    kernelarea <- list()

    for(i in 1:length(percentclasses)){
        one<-getverticeshr(hr, percent=percentclasses[i], whi = names(hr), unin = "m", unout="km2")
        k <- as.data.frame(one@data)
        k$stamp <- stamp
        k$timeunit <- timeunit
        k$size <-kernelindex[i]
        k$method <- method.name
        kernelarea[[kernelindex[i]]] <- k
```

```
    for(j in levels(one@data$id)){
        try1 <- assign(paste(path[[i]],j,"try", sep="_"), one[one@data$id %in% (j),])
        try2 <- spTransform(try1, CRS(workproj))
        writeOGR(try2, dsn=as.character(path[i]),
                 layer=paste(j,"subset_k",kernelindex[i],stamp,timeunit, sep="_"),
                 driver="ESRI Shapefile",check_exists=TRUE, overwrite_layer=TRUE)
        }
      }
      return(kernelarea)
  }

###   end of function    ##########################################################################################
#####     END OF SCRIPT      ##########################################################################################

  ###
  ### calculate_elevation.r
  ###
  ### extract and calculate elevation per individual per timeunit
  ###
  ##############################################################################################

  ### create list to store results
  elevlist <- list()
  ### for every timeunit (e.g. for every month)
  for(i in levels(timeunit)){
      ### intermediate storage of results
      animalslist <- list()
      ### subset for timeunit
      mon1 <- subset(animals, animals[,timeunit.colname] == i)
      ### drop unused levels
      mon2 <- droplevels(mon1)
      ### for every individual
      for(j in levels(mon2[,IDcolumnname])){
          ### subset individual
          datasub <- subset(mon2, mon2[,IDcolumnname] == j)
          ### create spatial object and project it
          coordinates(datasub) <-  coordcolumnnames
          proj4string(datasub) <- CRS(workproj)
          ### extract elevational information
          elev <- extract(aster, datasub)
```

```
    ### calculate mean and save in a list
    animalslist[[j]] <- mean(elev)
    ### create data frame and add additional information
    elevdf <- as.data.frame(do.call(rbind, animalslist))
    elevdf$id <- row.names(elevdf)
    elevdf$timeunit <- i
    elevdf$stamp <- stamp
  }
  elevlist[[i]] <- elevdf
}
######     END OF SCRIPT     ##########################################################################################


  ###
  ### create_buffers_around_hrcentres.r
  ###
  ### create buffers around home range centres and rasterize them
  ###
  ##############################################################################################################
  unloadNamespace("R.utils")
  unloadNamespace("R.oo")
  unloadNamespace("R.methodsS3")
  ###-----------------------------------------------------------------------------------------------------------
  ### subset  into the different years
  hr.centre.yearlist <- list()
  for(j in timespan){
    hr.centre.yearlist[[str_c("year_", j)]] <- subset(hr.centre_df, hr.centre_df$year == j)
  }
  ###-----------------------------------------------------------------------------------------------------------
  ### Note: single points need to be selected individually, otherwise buffer will intersect
  allpts <- list()
  ### create Spatial Points
  for(i in 1:length(hr.centre.yearlist)){
    pts <- list()
    sublist <- hr.centre.yearlist[[i]]
    for(j in 1:nrow(sublist)){
      coords <- cbind(sublist$hrcentre_x[j]  , sublist$hrcentre_y[j])
      pts[[str_c(sublist$timeunit[j],"_",sublist$id[j],"_",
                 sublist$method[j],sublist$size[j])]] <- SpatialPoints(coords,
                                                          proj4string=CRS(workproj))
    }
    allpts[[names(hr.centre.yearlist)[i]]] <- pts
```

```
}
###------------------------------------------------------------------------
### calculate buffers
all.buffers <- list()
for(j in 1:length(allpts)){
    pts <- allpts[[j]]
    buffers <- list()
    for(k in 1:length(buffersizes)){
        for(i in 1:length(pts)){
            buffers[[str_c("buff","_", buffersizes[[k]],"_", names(pts)[i])]] <- gBuffer(pts[[i]],
                byid=TRUE,id=NULL, width=buffersizes[[k]],capStyle="ROUND",quadsegs=50)
        }
    }
    all.buffers[[names(allpts)[j]]] <- buffers
}
save(all.buffers, file="buffers_around_hr.RData")
###------------------------------------------------------------------------
### subsample
setwd(RDatafolder)
load("buffers_around_hr.RData")
sub1 <- all.buffers[1][[1]][1:2]
all.buffers <- sub1

### count pixels
### results stored in "pixlist_buffers.RData" (RDatafolder)
###------------------------------------------------------------------------
years <- factor(timespan)
pixlist_all <- list()
### loop over years
for(i in levels(years)){
    print(i)
    ### get hrlist for respective year
    yearlist <- all.buffers[[paste("year", i, sep="_")]]
    ### loop over "list in list"
    pixlistcat7 <- list()
    ### loop over different kernels
    for(j in names(yearlist)){
        ### get VegetationMap for respective year
        rastername <- paste(name.veg.rast, i, ".tiff", sep="")
        landmap <- raster(paste(landusefolder, rastername, sep="/"))
        pixlistcat7[[paste("pixcat7",j, sep="_")]] <- extract(landmap, yearlist[[j]])
    }
```

```
    pixlist_all[[paste("pixlist_buffers", i , sep="_")]] <- pixlistcat7
    print("-----------------------------------------------------------------------")
  }
  ### save as RData file
  setwd(RDatafolder)
  save(pixlist_all, file="pixlist_buffers.RData")
  ###-----------------------------------------------------------------------
  ### get % veg in hr
  ###--------------------
  ### with NAs
  hr.count_all <- list()
  for(j in 1:length(pixlist_all)){
    pixlistcat7 <-  pixlist_all[[j]]
    anteile.7 <- list()
    na.anteile.7 <- list()
    tier.count.7 <- NA
    veg.count.7 <- list()
    tier.count.proz.7 <- list()
    for(i in 1:length(pixlistcat7)){
      pixsub <- unlist(pixlistcat7[[i]])
      pixsub[pixsub==0] = NA
      anteile.7[[names(pixlistcat7)[[i]]]] <- tabulate(pixsub, nbins=7)
      na.anteile.7[[names(pixlistcat7)[[i]]]] <- sum(is.na(pixsub))
      tier.count.7 <- c(anteile.7[[i]], na.anteile.7[[i]])
      names(tier.count.7) <- c(vegclass.names, "NA")
      veg.count.7[[names(pixlistcat7)[i]]] <- tier.count.7
      tier.count.proz.7[[names(pixlistcat7)[i]]] <- (100 * tier.count.7) / sum(tier.count.7)
    }
    hr.count_all[[str_c("buffer.count", "_", names(pixlist_all)[j])]] <- as.data.frame(do.call(rbind, tier.count.proz.7))
  }

  for(i in 1:length(hr.count_all)){
    namesplit <- as.data.frame(str_split_fixed(rownames(hr.count_all[[i]]), pattern="_",8))
    hr.count_all[[i]]$method <-  namesplit[,8]
    #    size.percent <- as.data.frame(str_split_fixed(namesplit[,3], pattern="hr",2))
    hr.count_all[[i]]$size <- paste(namesplit[,2], namesplit[,3],sep="_")
    hr.count_all[[i]]$id <- paste(namesplit[,5], namesplit[,6], namesplit[,7],sep="_")
    hr.count_all[[i]]$timeunit <- namesplit[,4]
    rownames(hr.count_all[[i]]) <- NULL
  }

  veg.df <- as.data.frame(do.call(rbind,hr.count_all))
```

```
rownames(veg.df)<- NULL
setwd(outputfolder) ; write.table(veg.df, "veg_in_hr_buffers.txt" ,row.names=FALSE,sep="\t")
###---------------------------------------------------------------------------------------------
### crop landmap to extent of single buffers (with respect to the year)
years <- factor(timespan)
### loop over years
for(i in levels(years)){
    mypat <- paste("year_",i,sep="")
    yearlist <- all.buffers[[mypat]]
    ### loop over buffers
    for(j in 1:length(yearlist)){
        hrcentresub <- yearlist[[j]]
        ### get VegetationMap for respective year
        rastername <- paste(name.veg.rast, i, ".tiff", sep="")
        landmap <- raster(paste(landusefolder, rastername, sep="/"))
        ### crop
        r <- crop(landmap, hrcentresub)
        namehr <- names(yearlist)[j]
        writeRaster(r, filename= paste(croppedBuffs, namehr, sep="/"),format="GTiff",overwrite=TRUE)
    }
}
###---------------------------------------------------------------------------------------------
### rasterize buffers
hrcentre.splist <- unlist(all.buffers, use.names=TRUE,recursive=FALSE)
newcol <- as.data.frame(str_split_fixed(names(hrcentre.splist), pattern="\\.",2))
names(hrcentre.splist) <- newcol[,2]
for(i in names(hrcentre.splist)){
    a <- hrcentre.splist[[i]]
    r <- raster(str_c(croppedBuffs, "/",i,".tif"))
    b <- rasterize(a, r, silent=TRUE)
    r4 <- overlay(r, b, fun=function(x,y){return(x*y)} )
    myname <- paste(i, "rast", sep="_")
    writeRaster(r4, filename= paste(rasterbuffs, myname,  sep="/"),format="GTiff",overwrite=TRUE)
}
###---------------------------------------------------------------------------------------------
#####     END OF SCRIPT     ####################################################################


###
### FragStats_session.r
###
### analysis of the landscape configuration within each home range
```

```
### will be performed in Fragstats
###
### For running the analysis in a batch-script, information about:
### - input filename (with path)
### - cellsize
### - background value (NAs are not accepted)
### - number of rows
### - number of columns
### - input data type
### are needed.
###
#########################################################################################################################

### creating batch
for(j in 1:length(targetbatch)){
    setwd(targetbatch[[j]])
    files <- list.files(pattern=".asc$")
    mynames <- files
    mynames <- as.data.frame(mynames)
    mynames$cellsize <- value.cellsize
    mynames$background <- value.background
    mynames$rows <- value.rows
    mynames$columns <- value.columns
    for(i in 1:nrow(mynames)){
        ### load .asc
        myrast <- raster(paste(targetbatch[j], files[i], sep="/"))
        mynames$asc_name[i] <- paste(myrast@layernames, ".asc", sep="")
        ### get information about rows and columns
        mynames$rows[i] <- myrast@nrows
        mynames$columns[i] <- myrast@ncols
    }
    ### column data type
    mynames$InputDataType <- value.InputDataType
    ### get path and file name
    mynames$mynames <- NA
    for(k in 1:nrow(mynames)){
        mynames$mynames[k] <- paste(inputasc[j], mynames$asc_name[k], sep="/")
    }
    ### change backslash to slash (for fragstats)
    mynames$mynames <- gsub("/", "\\\\", mynames$mynames)
    ### remove unnecessary  columns
    mynames$asc_name <- NULL
```

```
    ### get batch script
    setwd(fragbatchfolder[[j]])
    write.table(mynames, file=batchfilenames[[j]], sep=",", col.names=FALSE, row.names=FALSE,
                quote=FALSE)
  }
###-----------------------------------------------------------------------------------------

  ##########################################################################################
  ###    RUN FRAGSTATS                                                                   ###
  ##########################################################################################

  loadandinstall("R.utils")
  all.land <- list()
  for(i in 1:length(fragbatchfolder)){
      setwd(fragbatchfolder[[i]])
      files <- list.files()
      ### copy file, rename file, copy file to "fragstats_execution" folder = "Dummy Folder"
      file.copy(files[1], paste(files[1], "copy", sep="_"), overwrite=TRUE)
      file.rename(paste(files[1], "copy", sep="_"), batch.dummy)
      file.copy(batch.dummy, fragexefolder, overwrite=TRUE)
      setwd(fragbatchfolder[i])
      file.remove(batch.dummy)
      ### run fragstats from R
      system2(command=myfragstat, args = c(frag.parametrization))
      ### Note: Parametrization.frg must be manually created in Fragstats
      ### (specifies Input, Output and Indices)
      ### copy output to proper folder
      copyDirectory(from = outputDummyfolder, fragbatchfolder[i])
      ### create output table from fragstats results
      setwd(fragbatchfolder[i])
      a <- read.table(frag.outputfile, sep=",", header=TRUE)
      ### create one column with filename only
      split1 <- strsplit(as.character(a$LID), split="\\\\")
      split2 <- as.data.frame(do.call(rbind, split1))
      split3 <- as.data.frame(do.call(rbind,strsplit(as.character(split2[,ncol(split2)]),
                                                     split="\\_rast_to_ascii.asc")))

      a$filename <- split3[,1]
      a$LID <- NULL
      all.land[[i]] <- a
      ### delete content of "outputDummy" folder
      setwd(outputDummyfolder)
      remfile <- list.files()
```

```
  file.remove(remfile)
}

frag.all <- as.data.frame(do.call(rbind, all.land))
unloadNamespace("R.utils")
unloadNamespace("R.oo")
unloadNamespace("R.methodsS3")
##-------------------------------------------------------------------------------------------------
#####     END OF SCRIPT     ##########################################################################

  ###
  ### homerange_rasterisation.r
  ###
  ### rasterization of  home ranges
  ###
  ##########################################################################################################

  ### crop raster with hrsize
  ##-------------------------------
  years <- factor(timespan)
  ### loop over years
  for(i in levels(years)){
      ### get hrlist for respective year
      yearlist <- get(paste("hrlist", i, sep="_"))
      for(j in 1:length(yearlist)){
          hrlist <- yearlist[[j]]
          ### get VegetationMap for respective year
          rastername <- paste(name.veg.rast, i, ".tiff", sep="")
          landmap <- raster(paste(landusefolder, rastername, sep="/"))
          ### get single kernel / individuals
          for(k in 1:length(hrlist)){
              r <- crop(landmap, hrlist[[k]])
              namehr <- names(hrlist[k])
              writeRaster(r, filename= paste(allcropped[[j]], namehr, sep="/"),format="GTiff",
                          overwrite=TRUE)
          }
      }
  }
##-------------------------------------------------------------------------------------------------
### rasterize hr shape
##-------------------------------
```

```
for(j in 1:length(allcropped)){
    setwd(as.character(allcropped[[j]]))
    dsnbase=as.character(allcropped[[j]])
    dsnbase2=as.character(allrast[[j]])
    files <- list.files(pattern=".tif")
    hrlist <- hrlist_all[[j]]
    for(i in 1:length(hrlist)){
        a <- hrlist[[i]]
        r <- raster(paste(dsnbase, files[i], sep="/"))
        b <- rasterize(a, r, silent=TRUE)
        r4 <- overlay(r, b, fun=function(x,y){return(x*y)} )
        namehr <- names(hrlist[i])
        myname <- paste(namehr, "rast", sep="_")
        writeRaster(r4, filename= paste(dsnbase2, myname,  sep="/"),format="GTiff",overwrite=TRUE)
    }
}
###-----------------------------------------------------------------------------
#####     END OF SCRIPT     ##########################################################
#####


    ### hrcentre.r
    ###
    ### calculation of home range centre
    ###-----------------------------------

    hr.centre <- list()
    for(i in 1:length(percentclasses)){
        hr.centre[[method.id[i]]] <- as.data.frame(t(as.data.frame(sapply(hrlist[[i]],
        function(x) slot(x@polygons[[1]], "labpt")))))
        getyear <- as.data.frame(str_split_fixed(rownames(hr.centre[[i]]), pattern="_",5))
        hr.centre[[method.id[i]]]$year <- getyear[,4]
        row.names(hr.centre[[method.id[i]]]) <- NULL
        hr.centre[[method.id[i]]]$id <- name
        hr.centre[[method.id[i]]]$timeunit <- timeunit
        hr.centre[[method.id[i]]]$timestamp <- stamp
        hr.centre[[method.id[i]]]$size <- as.character(percentclasses[i])
        hr.centre[[method.id[[i]]]]$method <- method.name
        colnames(hr.centre[[method.id[i]]]) <- c("hrcentre_x", "hrcentre_y", "year", "id",
        "timeunit", "stamp", "size", "method")
    }
#####     END OF SCRIPT     ##########################################################
```

```
### hrlist.r
###
### create lists of all polygons (all kernel areas)
###-------------------------------------------------------

hrlist <- list()
for(i in 1:length(path)){
    hrlist_one <- list()
    setwd(as.character(path[i]))
    files <- list.files(pattern = ".shp$")
    split1 <- str_split(files, "_subset")
    split2 <- str_split(files, pattern= ".shp")
    split3 <- as.data.frame(do.call(rbind, split2))
    split4 <- str_split(split3[,1], paste(as.character(stamp), "_", sep=""))
    name <- sapply(split1, function(x) x[1])
    timeunit <- sapply(split4, function(x) x[2])
    dsnbase <- path[i]
    for(j in 1:length(files)){
        hrlist_one[[paste(method.id[i], stamp, name[j],
        timeunit[j], sep="_")]] <- readOGR(dsn = dsnbase, layer = split3[j,1], verbose=FALSE)
    }
    hrlist[[paste("hrlist", method.id[i], sep="_")]] <- hrlist_one
}
#####     END OF SCRIPT     ##########################################################

    ###
    ### ISOwek.r
    ###
    ### Inputs a date object, posix object, or 3 numbers and
    ### gives back the iso week.
    ### By Gustaf Rydevik, revised 2010
    ### source: http://quantitative-ecology.blogspot.com/2009/10/iso-week.html

    getweek<-function(Y,M=NULL,D=NULL){

        if(!class(Y)[1]%in% c("Date","POSIXt")) {
            date.posix<-strptime(paste(Y,M,D,sep="-"),"%Y-%m-%d")
        }
        if(class(Y)[1]%in% c("POSIXt","Date")){
```

```
  date.posix<-as.POSIXlt(Y)
  Y<-as.numeric(format(date.posix,
                       "%Y"))
  M<-as.numeric(format(date.posix,
                       "%m"))
  D<-as.numeric(format(date.posix,
                       "%d"))
}
LY<- (Y%%4==0 & !(Y%%100==0))|(Y%%400==0)
LY.prev<- ((Y-1)%%4==0 & !((Y-1)%%100==0))|((Y-1)%%400==0)
date.yday<-date.posix$yday+1
jan1.wday<-strptime(paste(Y,"01-01"
                          ,sep="-"),"%Y-%m-%d")$wday
jan1.wday<-ifelse(jan1.wday==0,7,
                  jan1.wday)
date.wday<-date.posix$wday
date.wday<-ifelse(date.wday==0,7,
                  date.wday)
####If the date is in the
####beginning, or end of the year,
### does it fall into a week of
###the previous or next year?

Yn<-ifelse(date.yday<=(8-jan1.wday)&jan1.wday>4,Y-1,
           ifelse(((365+LY-date.yday)<(4-date.wday)),Y+1,Y))

  ##Set the week differently if
  ##the date is in the
  ##beginning,middle or end of the
  ##year
  Wn<-ifelse(
      Yn==Y-1,
      ifelse((jan1.wday==5|(jan1.wday==6 &LY.prev)),53,52),
      ifelse(Yn==Y+1,1,(date.yday+(7-date.wday)+(jan1.wday-1))/7-(jan1.wday>4))
  )
  return(list(Year=Yn,ISOWeek=Wn))
}
######     END OF SCRIPT     ##########################################################################################

  ###
  ### libraries.r
```

```
###
### script to install and load libraries
###
################################################################################################

###---------------------------------------------------------------------------------------------
### R 2.13.1
### libraries
library("RODBC")         # 1.3-3
library("foreign")       # 0.8-44
library("rgdal")         # 0.7.1
library("sp")            # 0.9-91
library("maptools")      # 0.8-10
library("adehabitatHR")  # 0.3.2
library("proj4")         # 1.0-6
library("shapefiles")    # 0.6
library("memisc")        # 0.95-33
library("raster")        # 1.9-5
library("stringr")       # 0.5
library("spgrass6")      # 0.7-4
library("rgeos")         # 0.1-10
### R.utils causes problems with the package 'raster' and will only be loaded when needed
### and afterwards detached
### library(R.utils)            # 1.7.8
###---------------------------------------------------------------------------------------------
#####     END OF SCRIPT     ####################################################################


###
### SPGRASS6_session_convert_raster_to_ascii_with_and_without_header.r
###

################################################################################################
### GRASS session                                                                            ###
################################################################################################

### INITIALIZE GRASS
initGRASS(gisBase= mygrass, home=tempdir(), gisDbase=mygisDbase,
          location=mylocation,"",mapset=mymapset,override=TRUE)
###---------------------------------------------------------------------------------------------
### ASCII FILES WITHOUT HEADER
```

```
for(j in 1:length(targetfolder)){
    setwd(as.character(targetfolder[[j]]))
    files <- list.files(pattern=".tif")
    fn <- strsplit(files, split=".tif")
    fn2 <- as.data.frame(do.call(rbind, fn))
    dsnbase = targetfolder[[j]]
    for(i in 1:length(files)){
        ### load raster
        execGRASS("r.in.gdal", flags=c("e","overwrite", "quiet"),
                  parameters=list(input=paste(dsnbase, files[i],sep="/"), output= files[i]))
        ### create proper file name
        filename <- paste(fn2[i,1], "to_ascii.asc", sep="_")
        ### set proper g.region
        execGRASS("g.region", parameters = list(rast = files[i], res =myres))
        ### convert to ascii without header and write it to folder
        execGRASS("r.out.ascii", flags = c("i","h","quiet"), parameters = list(input=files[i],
            output = paste(output.asc.without.header[[j]], filename, sep="/"), null=mynullvalue))
    }
}
### remove loaded objects from grass mapset
execGRASS("g.mremove",flags=c("f", "b", "quiet"), parameters=list(rast="*"))
###-----------------------------------------------------------------------------------------

### ASCII FILES WITH HEADER

for(j in 1:length(targetfolder)){
    setwd(as.character(targetfolder[[j]]))
    files <- list.files(pattern=".tif")
    fn <- strsplit(files, split=".tif")
    fn2 <- as.data.frame(do.call(rbind, fn))
    dsnbase = targetfolder[[j]]
    for(i in 1:length(files)){
        ### load raster
        execGRASS("r.in.gdal", flags=c("e","overwrite", "quiet"),
                  parameters=list(input=paste(dsnbase, files[i],sep="/"),output= files[i]))
        ### create proper file name
        filename <- paste(fn2[i,1], "to_ascii.asc", sep="_")
        ### set proper g.region
        execGRASS("g.region", parameters = list(rast = files[i], res =myres))
        ### with header
        execGRASS("r.out.gdal", flags=c("quiet"), parameters=list(input=files[i], format="AAIGrid",
                  output = paste(output.asc.with.header[j], filename, sep="/"),
```

```
        type="Float64", nodata=as.numeric(mynullvalue)))
    }
}
### remove loaded objects from grass mapset
execGRASS("g.mremove",flags=c("f", "b", "quiet"), parameters=list(rast="*"))
###-------------------------------------------------------------------
#####     END OF SCRIPT     ##############################################

    ###
    ### veg_in_hr.r
    ###
    ### script to calculate the amount of vegetation within a home range
    ###
    ###################################################################################

    ###-------------------------------------------------------------------
    ### subset hrlist_all into the different years
    mylist <- list()
    for(j in timespan){
        for(i in names(hrlist_all)){
            mypat <- paste("_",j,"_", sep="")
            mylist[[i]] <- hrlist_all[[i]][str_detect(names(hrlist_all[[i]]), mypat)]
        }
        assign(paste("hrlist",j,sep="_"),mylist)
    }
###-------------------------------------------------------------------
### count pixels
### results stored in "pixlist_all.RData" (RDatafolder)
###-------------------------------------------------------------------
years <- factor(timespan)
pixlist_all <- list()
### loop over years
for(i in levels(years)){
    ### get hrlist for respective year
    yearlist <- get(paste("hrlist", i, sep="_"))
    ### loop over "list in list"
    pixlistcat7 <- list()
    ### loop over different kernels
    for(j in names(yearlist)){
        hrlist <- yearlist[[j]]
        ### get VegetationMap for respective year
```

```
    rastername <- paste(name.veg.rast, i, ".tiff", sep="")
    landmap <- raster(paste(landusefolder, rastername, sep="/"))

        ### get single kernel / individuals
        for(k in 1:length(hrlist)){
            pixlistcat7[[paste("pixcat7",names(hrlist)[k], sep="_")]] <- extract(landmap,
                                                                    hrlist[[k]])
        }
        pixlist1 <- pixlistcat7
    }
    pixlist_all[[paste("pixlist", i , sep="_")]] <- pixlist1
}
###-----------------------------------------------------------------------------
### save as RData file
setwd(RDatafolder)
save(pixlist_all, file=output.pixelcount)
###-----------------------------------------------------------------------------
### get % veg in hr
### with NAs
hr.count_all <- list()
for(j in 1:length(pixlist_all)){
    pixlistcat7 <-  pixlist_all[[j]]
    anteile.7 <- list()
    na.anteile.7 <- list()
    tier.count.7 <- NA
    veg.count.7 <- list()
    tier.count.proz.7 <- list()
    for(i in 1:length(pixlistcat7)){
        pixsub <- unlist(pixlistcat7[[i]])
        pixsub[pixsub==0] = NA
        anteile.7[[names(pixlistcat7)[[i]]]] <- tabulate(pixsub, nbins=7)
        na.anteile.7[[names(pixlistcat7)[[i]]]] <- sum(is.na(pixsub))
        tier.count.7 <- c(anteile.7[[i]], na.anteile.7[[i]])
        names(tier.count.7) <- c(vegclass.names, "NA")
        veg.count.7[[names(pixlistcat7)[i]]] <- tier.count.7
        tier.count.proz.7[[names(pixlistcat7)[i]]] <- (100 * tier.count.7) / sum(tier.count.7)
    }
    hr.count_all[[str_c("hr.count", "_", names(pixlist_all)[j])]] <- as.data.frame(
                                                    do.call(rbind, tier.count.proz.7))
}

for(i in 1:length(hr.count_all)){
```

```
  namesplit <- as.data.frame(str_split_fixed(rownames(hr.count_all[[i]]), pattern="_",8))
  hr.count_all[[i]]$method <- namesplit[,2]
  size.percent <- as.data.frame(str_split_fixed(namesplit[,3], pattern="hr",2))
  hr.count_all[[i]]$size <- size.percent[,2]
  hr.count_all[[i]]$id <- paste(namesplit[,5], namesplit[,6], namesplit[,7],sep="_")
  hr.count_all[[i]]$timeunit <- namesplit[,8]
  rownames(hr.count_all[[i]]) <- NULL
}

veg.df <- as.data.frame(do.call(rbind,hr.count_all))
veg.df[,"method"] <- str_replace(veg.df[,"method"],"k","kernel")
rownames(veg.df)<- NULL
##------------------------------------------------------------------------
#####    END OF SCRIPT    ##################################################
```

Chapter 3 - Adding structure to land cover – using fractional cover to study animal habitat selection

```
###
### prepare_landsat_data_for_input.r
###
library(raster)
library(rgdal)
library(stringr)

datafolder <- "F:/MB_remote/RS/fractionalCover/fCover_MB/01_data"

landsat_all <- str_c(datafolder, "/Landsat_without_clouds")
landsat_crop <- str_c(datafolder, "/Landsat_crop")

border <- readOGR(dsn="F:/MB_remote/RS/GeodatenShare/shapes", layer="Nationalpark_BW_Sumava_UTM")

setwd(landsat_all)
all.files <- list.files()
### create extent
myext <- extent(border)+15000
### crop to extent and write to folder
for(i in 1:length(all.files)){
  a <- raster(str_c(landsat_all, "/",all.files[i]))
  b <- crop(a, myext)
  writeRaster(b, filename=str_c(landsat_crop, "/", names(a), "_crop"), format="GTiff", overwrite=TRUE)
}

### create stack and save
setwd(landsat_crop)
files <- list.files()
inImage_Landsat <- stack(files)
save(inImage_Landsat, file=str_c(datafolder, "/inImage_Landsat.RData"))
###-------------------------------------------------------------------------------------
#####     END OF SCRIPT     ###############################################################


###
### percentCoverResample_class1_forest.r
###
```

```
#########################################################################################################
#Load libraries
library(raster)
library(rgdal)
library(stringr)

### paths
mainfolder <- "F:/MB_remote/RS/fractionalCover/fCover_MB"
datafolder <- str_c(mainfolder, "/01_data")
codefolder <- str_c(mainfolder, "/02_code")
outputfolder <- str_c(mainfolder, "/03_results")

#########################################   SET VARIABLES HERE  ###########################################
# Number of samples to be selected
numSamps <-10000
# Name and path for the input classified image

### create inClassImage by rasterization
myshp <- readOGR(dsn=datafolder, layer="landuseBWCZ_cat3")
myrast <- raster(str_c(datafolder, "/reference_raster_rasterization/VegetationMap_2000.tiff"))
inputrast <- rasterize(myshp, myrast, field=myshp@data$num_cat3, fun="max", progress="text")
writeRaster(inputrast, str_c(datafolder, "/01_data/inputrast_fcov.tif", format="GTiff"))

inClassImage <- str_c(datafolder, "/inputrast_fcov.tif")
# Name and path for the input image that will be used for predictions
inPredImage <- str_c(datafolder, "/Landsat_crop")
# No data value for the prediction image (Landsat)
ndPred <- 255
# Class numbers that will be mapped using the following scheme:
# 0 = no data such as background, clouds and shadow
# 1 = class for which percent cover is being calculated
# 2 = all other land cover classes
fromVals <- c(1,2,3)
toVals <-   c(1,2,2)
### 1 = forest
### 2 = grassland
### 3 = other
# Threshold for no-data processing
noDataPct <- 0.1
# Output file path and name
outFile <- str_c(outputfolder, "/outSamples_class1_forest.csv")
```

```
##########################################################################################

    # Start processing
    print("Set variables and start processing")
    startTime <- Sys.time()
    cat("Start time", format(startTime),"\n")

    # Load the classified image
    classImage <- raster(inClassImage)

    # Load the first band of the image that will be used for predicitons.
    # predImage <- raster(inPredImage, band=1)
    setwd(inPredImage) ; files <- list.files()
    predImage <- raster(str_c(inPredImage, "/", files[1]))

    # Calculate the resolution of the two images
    predImageRes <- res(predImage)[1]
    classImageRes <- res(classImage)[1]
    # Calculate the distance from the center of a raster cell to the edge - assuming a square pixel
    halfCell <- predImageRes/2

    # Check to make sure fromVals and toVals are the same length
    if (length(fromVals) != length(toVals)) {
        stop("fromVals and toVals must have the same number of values \n\n", call.=FALSE)
    }

    # Select the class values that will be used to create the percent cover map
    percentCoverValues <- fromVals[toVals == 1]

    # Calculate the common extent to ensure sample cover both images
    commonExt <- intersect(extent(predImage), extent(classImage))


    # Select random samples from the prediciton image
    cat("\nSelecting", numSamps, "samples from the prediction image\n")

    ### add crop to limit sample points to extent
    sampleCells <- sampleRandom(crop(predImage,commonExt), size=numSamps, xy=TRUE,
                                ext=commonExt, na.rm=TRUE)

    ### change argument sp to cells again
    sampleCells <- sampleRandom(crop(predImage,commonExt), size=numSamps, cells=TRUE,
```

```
            ext=commonExt, na.rm=TRUE)[,1]
lenSampCells <- nrow(sampleCells)
# Create the matrix that will hold the output data and label the columns
outputMatrix <- matrix(nrow=lenSampCells, ncol=3)
colnames(outputMatrix) <- c("X", "Y", "pct_cover")
fromTo <- data.frame(from=fromVals, to=as.integer(toVals==1))

# Get the pixels values from the classified image that fit inside the selected course-resolution pixel
cat("Calculating percent cover values for the output matrix\n\n")
for (i in 1:lenSampCells) {
    # Get the x and y coordinate from the center of the predition image pixel indicated by the sample point
    centerCoords <- sampleCells[i,1:2]
    # Insert x and y coordinate into the output matrix
    outputMatrix[i,1:2] <- c(centerCoords)
    # Calculate the extent of the selected prediction image pixel by adding/subtracting
    # half the resolution of the pixel
    pixelExt <- extent(centerCoords[1] - halfCell, centerCoords[1] + halfCell,
                       centerCoords[2] - halfCell, centerCoords[2] + halfCell)
    # Get the cell numbers of all the classified image pixels that fall inside the extent
    # of the selected prediction image pixel
    classCellNumbers <- cellsFromExtent(classImage, pixelExt)
    # Get the class number of all of the selected classified image pixels
    classCellValues <- extract(classImage, classCellNumbers)
    # Convert classCellValues no-data pixels (0 in toVals) to NA
    classCellValues[classCellValues %in% fromVals[toVals==0]] <- NA
    # If the number of no-data pixels from the classified image is less than 'noDataPct'
    # then calculate the percent cover
    # otherwise output NA
    if (sum(is.na(classCellValues))/length(classCellValues) < noDataPct) {
        outputMatrix[i,3] <- sum(!is.na(match(classCellValues,
                                 fromVals[toVals==1])))/length(classCellNumbers)
    } else {
        outputMatrix[i,3] <- NA
    }
    if (i %% 25 == 0)    cat("Processing sample", i, "of", lenSampCells, "\r")
}

# Write out the non-NA values in the output matrix to a CSV file
write.csv(outputMatrix[which(!is.na(outputMatrix[,3])),], outFile, row.names=FALSE)
# Calculate processing time
timeDiff <- Sys.time() - startTime
cat("Processing time", format(timeDiff), "\n")
```

```
###------------------------------------------------------------------------
#####     END OF SCRIPT     ###################################################################################

    ###
    ### rf_percentCover_class1_forest.r
    ###
    ##########################################################################################################

    #Load libraries
    require(maptools)
    require(sp)
    require(randomForest)
    require(raster)
    require(rgdal)
    library(stringr)

    ### paths
    mainfolder <- "F:/MB_remote/RS/fractionalCover/fCover_MB"
    datafolder <- str_c(mainfolder, "/01_data")
    codefolder <- str_c(mainfolder, "/02_code")
    outputfolder <- str_c(mainfolder, "/03_results")

    ##################################    SET VARIABLES HERE  ##############################################
    # The CSV file containing X, Y, and percent cover point data created by
    # the percentCoverResample.R scriptt.
    pointData <- str_c(outputfolder, "/outSamples_class1_forest.csv")

    load(str_c(datafolder, "/inImage_Landsat.RData"))
    inImage <- inImage_Landsat

    # Name and path of the output GeoTiff image
    outImage <- str_c(outputfolder,  "/out_percent_class1_forest.tif")
    # No data value for satellite image
    nd <- 255
    ##########################################################################################################

    # Start processing
    print("Set variables and start processing")
    startTime <- Sys.time()
    cat("Start time", format(startTime),"\n")
```

```
pointTable <- read.csv(pointData, header=TRUE)
xy <- SpatialPoints(pointTable[,1:2])
response <- as.numeric(pointTable[,3])

# Load the moderate resolution image
# satImage <- stack(inImage)
satImage <- inImage_Landsat
for (b in 1:nlayers(satImage)) { NAvalue(satImage@layers[[b]]) <- nd }

# Get pixel DNs from the input image for each sample point
print("Getting the pixel values under each point")
trainvals <- cbind(response, extract(satImage, xy))

# Remove NA values from trainvals
trainvals_no_na <- na.omit(trainvals)

# Run Random Forest
print("Starting to calculate random forest object")
randfor <- randomForest(response ~. , data=trainvals_no_na, ntree=1000)
# Start predictions
print("Starting predictions")
predict(satImage, randfor, filename=outImage, progress='text', format='GTiff',
        datatype='FLT4S', type='response', overwrite=TRUE)

# Calculate processing time
timeDiff <- Sys.time() - startTime
cat("Processing time", format(timeDiff), "\n")
##-----------------------------------------------------------------------
#####     END OF SCRIPT     ###########################################
##-----------------------------------------------------------------------

###
### CorrectRF_Result_forest.r
###
#########################################################################

# When used for regression the random forests algorithm will overestimate low values
# and underestimate high values. This script adjusts this effect by calculating
# regression coefficients using the response variable from the training data and the
# corresponding predictor variables from the image output using X/Y coordinates provided
# by the input file. A new adjusted image image is output after applying gain and offset
```

```
# (slope and intercept) values to the original predicted image values. The user can
# select the type of regression to apply and set the minimum and maximum values for the
# output image. This is useful when processing percent cover images when the valid range
# is between 0 and 1.
#
# Set the variables below in the "SET VARIABLES HERE" section of the script.
#
# This script was written by Ned Horning [horning@amnh.org]
# Support for writing and maintaining this script comes from The John D. and
# Catherine T. MacArthur Foundation.
#
# This script is free software; you can redistribute it and/or modify it under the
# terms of the GNU General Public License as published by the Free Software Foundation
# either version 2 of the License, or ( at your option ) any later version.
#
##############################################################################
#Load libraries
require(maptools)
require(sp)
require(randomForest)
require(raster)
require(rgdal)

#############################    SET VARIABLES HERE  ###########################################
# Enter a 1 if the response variable training data are in a .dbf file or 2 if the data are in a
# CSV format  with a header
fileType <- 2
# The CSV or DBF file containing X, Y, and response variable (i.e., biomass, % cover...) data.
pointData <- paste0(mainfolder, "/03_results/outSamples_class1_forest.csv")
# Name and path for the input predicted image
inImage <- paste0(mainfolder, "/03_results/out_percent_class1_forest.tif")
# Name and path of the output adjusted image
outImage <-paste0(mainfolder, "03_results/out_percent_class1_forest_Corrected.tif")
# No-data value for the input image
nd <- 255
# Enter EITHER the name (case sensitive and in quotes) or the column number of the
# field containing X coordinates
x_coord <- "X"
# Enter EITHER the name (case sensitive and in quotes) or the column number of the
# field containing Y coordinates
y_coord <- "Y"
# Enter EITHER the name (case sensitive and in quotes) or the column number of the
```

```
# field containing the response variable values
responseVar <- "pct_cover"
# Minimum valid output value
minValue <- 0
# Maximum valid output value
maxValue <- 1
# Number of points to be randomly sampled. Enter -1 to use all sample points. It
# may be necessary to use a subset of the sample points to avoid memory problems.
numSamps <- -1
#
# Type of regression to be applied- 1 = linear, 2 = linear with intercept of 0
regType = 1
#
# Display regression graphs and compare different models (TRUE or FALSE)?
# Solid line shows regression for Theil-Sen Siegel repeated medians
# Dashed line shows regression for linear regression
# Dotted line shows regression for linear regression with intercept of 0
# Display a graph?
dispGraphs = TRUE
##############################################################################
# Start processing
cat("Set variables and start processing\n")
startTime <- Sys.time()
cat("Start time", format(startTime),"\n")

# Read the data in the dbf or CSV file and create a table of XY values and another
# with the response variable values
if (fileType==1) {
    xy <- read.dbf(pointData)[,c(x_coord, y_coord)]
    responseVar <- as.numeric(read.dbf(pointData)[,responseVar])
} else if (fileType==2) {
    pointTable <- read.table(pointData, header=TRUE, sep=",")
    xy <- SpatialPoints(pointTable[,c(x_coord, y_coord)])
    responseVar <- as.numeric(pointTable[,responseVar])
}

# Sample xy and responce if numSamps is not -1
#
if (numSamps != -1) {
    if (numSamps > length(xy[,1])) {
        cat("\n\n**********************************************************************")
        cat("\nThe variable numSamps is greater than the total number of
```

```
    points available to be randomly sampled\n")
  cat("Please change numSamps to be smaller than", length(xy[,1]), "which is
      the total number of sample points.\n\n")
  stop("Change the value for numSamps and then restart the script\n\n", call.=FALSE)
  }
  sampIdx = sample(seq(1, length(xy[,1])), size=numSamps)
  xy = xy[sampIdx,]
  responseVar = responseVar[sampIdx]
}

# Load the input predected image then flag all no-data values (nd) so they are not processed
predImage <- raster(inImage)
NAvalue(predImage) <- nd

# Get pixel responseVar values from the image under each sample point and create a table with
# observed and predicted responseVar values
cat("Getting the pixel values under each point\n")
samples <- cbind(responseVar, extract(predImage, xy))

# Remove NA values from trainvals table created above
samples <- na.omit(samples)

# Calculate blocksize for image writing at the end of the script
bs <- blockSize(predImage)

# Get output data type from the input image
dataType <- dataType(predImage)

samplesX <- samples[,2] # Predicted values from the input image
samplesY <- samples[,1] # Observed values from training data

cat("Plot graphs\n")
if (regType==1) {
  fit <- lm(samplesY ~ samplesX)
  slope <- fit$coefficients[2]
  intercept <- fit$coefficients[1]
  if (dispGraphs) {
    plot(samplesX, samplesY, xlab="Predicted", ylab="Observed")
      abline(fit, lty=2, col="red")
      abline(lm(samplesY ~ samplesX+0), lty=3, col="red")
  }
} else if (regType==2) {
```

```
    fit <- lm(samplesY ~ samplesX+0)
    slope <- fit$coefficients[1]
    intercept <- 0
    if (dispGraphs) {
        plot(samplesX, samplesY, xlab="Predicted", ylab="Observed")
        abline(fit, lty=3, col="red")
        abline(lm(samplesY ~ samplesX), lty=2, col="red")
    }
} else {
    cat("\n   The variable regType must be an integer between 1 and 3 \n\nEnter Q to quit\n\n")
    browser()
}

print(fit)
cat("Writing output image\n")
#Create output image and start writing to it.
img.out <- raster(predImage)
img.out <- writeStart(img.out, outImage, overwrite=TRUE, datatype=dataType)

#Loop over blocks of the image and write the adjusted values

for (i in 1:bs$n) {
    cat("Processing block", i, "of", bs$n, "\r")
    img <- getValues(predImage, row=bs$row[i], nrows=bs$nrows[i])
    img.pred <- img * slope + intercept
    # Set the no data value to the default value for the output image
    img.pred[is.na(img.pred) == TRUE] <- nd
    img.pred[img.pred < minValue] <- minValue
    img.pred[img.pred > maxValue] <- maxValue
    writeValues(img.out, img.pred, bs$row[i])
}
cat("\n")
#Finish saving and close connection to image.
img.out <- writeStop(img.out)
#
# Calculate processing time
timeDiff <- Sys.time() - startTime
cat("Processing time", format(timeDiff), "\n")
###------------------------------------------------------------------------
#####     END OF SCRIPT     ##################################################
```

```
### please see chapter two for home range calculation

    ###
    ### calculation_of_texture.r
    ###

    library(raster)
    library(stringr)
    library(EBImage)

    eb.all <- list()
    for(x in 1:length(allrast)){
      setwd(allrast[[x]])
        files <- list.files()
        eb <- list()
        for(f in 1:length(files)){
            print(f)
            if(f == 1178){ next }
            r <- raster(files[f])
            m <- !(is.na(r))
            r <- as.matrix(r)
            m <- as.matrix(m) * 1
            r[is.na(r)] <- 0
            res <- computeFeatures.haralick(m,r)
            eb[[files[f]]] <- res
        }
        eb.one <- as.data.frame(do.call(rbind, eb))
        eb.one$names <- names(eb)
        eb.all[[x]] <- eb.one
    }
    eb.df <- as.data.frame(do.call(rbind, eb.all))
    save(eb.df, file=str_c(RDatafolder,"/eb_df_", stamp, "_rast.RData"))
    ###-------------------------------------------------------------------------
    #####     END OF SCRIPT     ##################################################
```