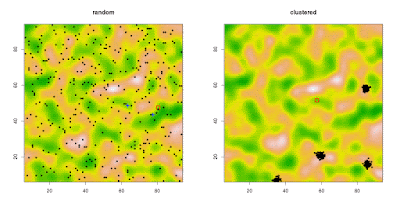
Hi there!

 In 2010, when I was studying miniseries called [Great Migrations](https://en.wikipedia.org/wiki/Great_Migrations)([National Geographic](http://www.nationalgeographic.com.au/tv/great-migrations/)). Their episodes talk about awesome migrations of animals around the globe. One of these chapters is “Great Migrations: Science Of Migrations”. It shows how scientists study the patterns and processes of animal migration. Through simulations, I saw how wildebeests maintaining a clustered movement pattern can more effectively avoid to be hunted by a virtual lion. I have to confess that each time I see those images, I think that I would be completely fulfilled with this kind of jobs.

[](https://i2.wp.com/1.bp.blogspot.com/-s-sAxqBawaQ/W1Ysm0xr77I/AAAAAAAAARs/IyxYot7SI68e-df7s1mb9SOrqxezqQj8ACLcBGAs/s1600/random_cluster.gif?ssl=1)

I modified a little bit my “go” function to reproduce the experiment shown by John. This is a small tribute to this documentary series and to all those people who inspire and motivate us and arouse our curiosity, as John.

You can find the code to perform these simulations with R below.

That is all, enjoy!

# First, we prepare packages and raster layers to be used later. We also load   
 # the species class to define the hunter inital location and his environmental   
 # optimum (grass cover in this case), as in previous posts.   
   
 library (raster)   
 library(dismo)   
   
 gc <- raster(nrows = 100, ncols = 100, xmn = 0, xmx = 100, ymn = 0, ymx = 100)   
 gc[] <- runif(10000, -80, 180)   
 gc <- focal(gc, w = matrix(1, 5, 5), mean)   
 gc <- focal(gc, w = matrix(1, 5, 5), mean)   
   
 species <- setClass("species", slots = c(x = "numeric", y = "numeric",   
 opt = "numeric"))   
   
 # Then we will load the hunt function. This is pretty similar to my previous   
 # "go" function. However, I added a "preys" attribute, a set of locations   
 # of available preys. After each movement of the hunter, the function checks   
 # if the hunter is in the same cell as any prey. If yes, it records this   
 # location and delete this prey from the available preys. The output is a 4   
 # column dataframe: the first two are the location of the hunter at each   
 # iteration. The last two show if there is any capture at this iteration.   
   
 hunt <- function (sp, env, n, sigma, theta\_x, alpha\_x, theta\_y, alpha\_y, preys) {   
 track <- data.frame() #create an empty data.frame to store each step coordinates   
 track[1,1] <- sp@x #the first position will be the initial position of the specimen defined by the species class   
 track[1,2] <- sp@y #the first position will be the initial position of the specimen defined by the species class   
 hunted <- data.frame()#create an empty data.frame to store hunted preys   
   
   
 for (step in 2:n) {   
   
 #update preys locations   
   
 prey\_cells <- cellFromXY(env, as.matrix(preys + step))   
   
 #First, the function searches in the adjacent cells what have the value that best fits with the species requirements   
 lon\_candidate<--9999   
 lat\_candidate<--9999   
 #randomization/atraction routine   
 while ( is.na(extract(env, matrix(c(lon\_candidate,lat\_candidate),1,2)))) {   
 lon\_candidate <- track[step-1,1]+ (sigma \* rnorm(1)) + (alpha\_x \* ( theta\_x - track[step-1,1]))   
 lat\_candidate <- track[step-1,2]+ (sigma \* rnorm(1)) + (alpha\_y \* ( theta\_y - track[step-1,2]))   
 }   
   
 neig <- adjacent(env,   
 cellFromXY(env, matrix(c(lon\_candidate,   
 lat\_candidate), 1,2)),   
 directions=8, pairs=FALSE )   
   
 options <- data.frame()   
 for (i in 1:length(neig)){   
 options[i,1]<-neig[i]   
 options[i,2]<- sp@opt - env[neig[i]]   
 }   
 option <- c(na.omit(options[abs(options$V2) == min(abs(na.omit(options$V2))), 1 ]),   
 na.omit(options[abs(options$V2) == min(abs(na.omit(options$V2))), 1 ]))   
   
 new\_cell <- sample(option,1)   
 new\_coords <- xyFromCell(env,new\_cell)   
   
 track[step,1] <- new\_coords[1]   
 track[step,2] <- new\_coords[2]   
   
 if (new\_cell %in% prey\_cells) {   
 hunted[step,1] <- new\_coords[1]   
 hunted[step,2] <- new\_coords[2]   
 preys <- preys[-match(new\_cell,prey\_cells),]   
 }   
 else {   
 hunted[step,1] <- NA   
 hunted[step,2] <- NA   
 }   
 }   
 return(cbind(track,hunted))   
 }   
   
 # Later we define our predator, in this case, a lion! The inital possition will   
 # be at the middle of the space (50,50). We will also select the environmental   
 # optimum of our lion, 60% of grass cover (our lion hates bare ground xD)   
   
 lion <- species(x= 50, y =50, opt= 60)   
   
 # Now, we will set the locations of a bunch of wildebeests using "randomPoints"   
 # from "dismo" package. As our wildebeests will be running around, we will   
 # initially locate them at the bottom left.   
   
 r <- raster(nrows=500, ncols=500, xmn=-500, xmx=0, ymn=-500,ymx=0)   
 wb <- randomPoints(r, 10000)   
   
 # Here we have our wildebeests ready to run!   
   
 plot(wb)   
   
 # Let start the hunting!   
   
 hunting <- hunt(lion, gc, 600, 2, 0, 0, 0, 0, wb)   
   
 # In "hunting" object is stored each location of our lion during the 600   
 # iterations, as well as each time she catches a wildebeest. To see only the   
 # catches, you can use:   
   
 na.omit(hunting)   
   
 # Pretty awesome, right? But, what happens if we group our wildebeests in   
 # clusters? To do it, we will define another small function that creates   
 # clusters of points:   
   
 clus <- function(n,m1,m2){   
 df<-data.frame()   
 for (i in 1:n){   
   
 a <- rnorm(100,m1[i],1)   
 b <- rnorm(100,m2[i],1)   
 df <- rbind(df,data.frame(a,b))   
 }   
   
 return(df)   
 }   
   
 wb\_clustered <- clus(100, runif(100,-500,0),runif(100,-500,0))   
   
 # Look at the wildebeests locations now!   
   
 plot(wb\_clustered)   
   
 # Ok, let's go again!   
   
 hunting\_clustered <- hunt(lion, gc, 600, 2, 0, 0, 0, 0, wb\_clustered)   
   
 # Let's chek how many catches he have now:   
   
 na.omit(hunting\_clustered)   
   
 # And comparing both strategies...   
   
 nrow(na.omit(hunting))   
 nrow(na.omit(hunting\_clustered))   
   
 # ...looks like wildebeests should stay together!!   
 # You can run these simulations n times to see if the results are robust.   
   
 # Finally we can print each iteration to make a cool gif    
   
 for (i in 1:600) {   
 id <- sprintf("%03d", i)   
 png(paste("lion1\_",id,".png", sep=""), width=1300, height=690, units="px",   
 pointsize = 16)   
   
 par(mfrow=c(1,2))   
   
 plot(gc, xlim = c(6,94), ylim = c(6,94), main="random", legend=FALSE)   
 points(hunting[i,1:2], cex=1.5, col="red", pch = 0, lwd =3)   
 points(hunting[1:i,3:4], cex=1, col="blue", pch = 3, lwd =3)   
 points (gnu +i, pch = 20)   
   
 plot(gc, xlim = c(6,94), ylim = c(6,94), main="clustered", legend=FALSE)   
 points(hunting\_clustered[i,1:2], cex=1.5, col="red", pch = 0, lwd =3)   
 points(hunting\_clustered[1:i,3:4], cex=1, col="blue", pch = 3, lwd =3)   
 points (gnu\_clustered +i, pch = 20)   
   
 dev.off()   
 }