# SEPARATING MORTALITY AND EMIGRATION: MODELLING SPACE USE, DISPERSAL AND SURVIVAL WITH ROBUST-DESIGN SPATIAL-CAPTURE-RECAPTURE DATA

Torbjørn Ergon (1)\* and Beth Gardner (2)

(1): Centre for Ecological and Evolutionary Synthesis, Department of Biosciences, University of Oslo, P.O. Box 1066 Blindern, N-0316 Oslo, Norway

(2): Department of Forestry and Environmental Resources, North Carolina State University, Campus Box 7646, Raleigh, NC 27695-7646, USA

\* Corresponding author: [torbjorn.ergon@bio.uio.no](mailto:torbjorn.ergon@bio.uio.no)

## Appendix S3: Example R-script, including JAGS model specification

###################################################################

### JAGS MODEL SPECIFICATION - Spatial robust design model with ###

### (zero-inflated) exponential dispersal distribution ###

###################################################################

#

# DATA:

# - N[1]: Number of individuals that are only seen in the last primary session

# or the session they are censored (these individuals must be sorted

# first in the data)

# - N[2]: Total number of individuals

# - n.prim: Number of primary sessions

# - dt[k]: Length of interval k

# - tod[k,j]: Category of trapping session k,j; two categories (e.g., time-of-day)

# - first[i]: First primary session of individual i

# - K[i]: Last primary session for individual i (allows censoring)

# - J[i,k]: Last secondary session for individual i in primary session k

# (allows censoring)

# - gr[i]: Group (sex) of individual i

# - R: Number of traps

# - X[r,]: Location of trap r = 1..R (coordinate)

# - H[i,j,k]: Trap index for capture of individual i in secondary session j

# within primary session k. 1 = not captured, other values are r+1

# - Ones[i,j,k]: An array of all ones used in the "Bernoulli-trick" (see

# OpenBUGS user manual) in the observation likelihood. This saves

# computation time as it is not necessary to compute the complete

# capture probabilities for every individual\*trap\*session (it is

# sufficient to compute the g-variable for every

# ind\*trap\*primary)

# - xlow[i]: Lower bound in uniform prior for first x-location of individual i

# - xupp[i]: Upper bound in uniform prior for first x-location of individual i

# - ylow[i]: Lower bound in uniform prior for first y-location of individual i

# - yupp[i]: Upper bound in uniform prior for first y-location of individual i

#

# PARAMETERS:

# - kappa, sigma and lambda: Space use and recapture probability parameters

# (eq. 5 in paper)

# - beta: additive effects on log(lambda):

# - beta[1]: effect of tod==2

# - beta[2]: effect of gr==2

# - Phi[gr[i]]: Survival for group gr[i] per time-unit

# - phi[gr[i],k]: Survival probability for individual i belonging to group (sex)

# gr[i] over the k'th interval given that the individual is

# alive at the beginning of the interval.

# - dmean[gr[i]]: Mean dispersal distance (exponential distribution) given

# dispersal for group gr[i]

# - Psi[gr[i]]: Probability of not emigrating for group gr[i] per time-unit

# (only for zero-inflated model, commented out)

# - psi[gr[i],k]: Probability of dispersal (1-zero-inflation probability) for

# group gr[i] in interval k (only for zero-inflated model,

# commented out)

# - d.offset[gr[i]]: Minimum dispersal distance given dispersal for group gr[i]

# (only for zero-inflated model, commented out)

# STATE VARIABLES:

# - z[i,k]: Alive state of individual i in primary session k (1=alive, 0=dead)

# - S[i,,k]: Centre of activity of individual i in primary session k (x- and

# y-coordinate)

# - u[i,k]: Dispersal state for individual i in interval k (1=dispersal,

# 0=no dispersal)(only for zero-inflated model, commented out)

# Writing the model specification to a text file:

sink("RD-SCR.Exp.txt")

cat("

model{

## PRIORS AND CONSTRAINTS: ##

# Space use and recapture probability parameters:

for(sex in 1:2){

kappa[sex] ~ dunif(0,50)

sigma[sex] ~ dunif(0.1,20)

}

for(sex in 1:2){

for(TOD in 1:2){

lambda[TOD, sex] <- exp(log.lambda0) \* pow(beta[1],(TOD-1)) \* pow(beta[2],(sex-1))

}

}

PL ~ dunif(0.01,0.99)

log.lambda0 <- log(-log(1-PL))

beta[1] ~ dunif(0.1,10)

beta[2] ~ dunif(0.1,10)

# Survival parameters:

for(sex in 1:2){

Phi[sex] ~ dunif(0,1)

for(k in 1:(n.prim-1)){

phi[sex,k] <- pow(Phi[sex], dt[k])

}

}

# Dispersal parameters:

for(sex in 1:2){

dmean[sex] ~ dunif(0,100)

dlambda[sex] <- 1/dmean[sex]

# For zero-inflated model:

# d.offset[sex] ~ dunif(5,100)

# Psi0[sex] ~ dunif(0,1)

# for(k in 1:(n.prim-1)){

# psi[sex,k] <- 1 - pow(Psi0[sex], dt[k])

# }

}

## MODEL: ##

# Loop over individuals that are only seen in the last primary session or the

# session they are censored

for(i in 1:N[1]){

z[i,first[i]] ~ dbern(1)

S[i,1,first[i]] ~ dunif(xlow[i], xupp[i]) # Prior for the first x coordinate

S[i,2,first[i]] ~ dunif(ylow[i], yupp[i]) # Prior for the first y coordinate

g[i,first[i],1] <- 0

for(r in 1:R){ # trap

D[i,r,first[i]] <- sqrt(pow(S[i,1,first[i]]-X[r,1],2) + pow(S[i,2,first[i]]-X[r,2],2))

g[i,first[i],r+1] <- exp(-pow(D[i,r,first[i]]/sigma[gr[i]], kappa[gr[i]])) # Trap exposure

}

G[i,first[i]] <- sum(g[i,first[i],]) # Total trap exposure

for(j in 1:J[i,first[i]]){

P[i,j,first[i]] <- 1 - exp(-lambda[tod[first[i],j],gr[i]]\*G[i,first[i]]) # Probability of being captured

PI[i,first[i],j] <- step(H[i,j,first[i]]-2)\*(g[i,first[i],H[i,j,first[i]]]/(G[i,first[i]]+ 0.000000001))\*P[i,j,first[i]] + (1-step(H[i,j,first[i]]-2))\*(1-P[i,j,first[i]])

Ones[i,j,first[i]] ~ dbern(PI[i,first[i],j])

}

}

# Loop over all other individuals

for(i in (N[1]+1):N[2]){

z[i,first[i]] ~ dbern(1)

S[i,1,first[i]] ~ dunif(xlow[i], xupp[i]) # Prior for the first x coordinate

S[i,2,first[i]] ~ dunif(ylow[i], yupp[i]) # Prior for the first y coordinate

# First primary session:

g[i,first[i],1] <- 0

for(r in 1:R){ # trap

D[i,r,first[i]] <- sqrt(pow(S[i,1,first[i]]-X[r,1],2) + pow(S[i,2,first[i]]-X[r,2],2))

g[i,first[i],r+1] <- exp(-pow(D[i,r,first[i]]/sigma[gr[i]], kappa[gr[i]])) # Trap exposure

}

G[i,first[i]] <- sum(g[i,first[i],]) # Total trap exposure

for(j in 1:J[i,first[i]]){

P[i,j,first[i]] <- 1 - exp(-lambda[tod[first[i],j],gr[i]]\*G[i,first[i]]) # Probability of being captured

PI[i,first[i],j] <- step(H[i,j,first[i]]-2)\*(g[i,first[i],H[i,j,first[i]]]/(G[i,first[i]]+ 0.000000001))\*P[i,j,first[i]] + (1-step(H[i,j,first[i]]-2))\*(1-P[i,j,first[i]])

Ones[i,j,first[i]] ~ dbern(PI[i,first[i],j])

}

## Later primary sessions

for(k in (first[i]+1):K[i]){ # primary session

theta[i,k-1] ~ dunif(-3.141593,3.141593) # Prior for dispersal direction

z[i,k] ~ dbern(Palive[i,k-1])

Palive[i,k-1] <- z[i,k-1]\*phi[gr[i],k-1] # Pr(alive in primary session k) gr[i] = sex

d[i,k-1] ~ dexp(dlambda[gr[i]])

# For zero-inflated model, replace line above with:

#u[i,k-1] ~ dbern(psi[gr[i],k-1])

#dd[i,k-1] ~ dexp(dlambda[gr[i]])

#d[i,k-1] <- u[i,k-1]\*(d.offset[gr[i]] + dd[i,k-1])

S[i,1,k] <- S[i,1,k-1] + d[i,k-1]\*cos(theta[i,k-1])

S[i,2,k] <- S[i,2,k-1] + d[i,k-1]\*sin(theta[i,k-1])

g[i,k,1] <- 0

for(r in 1:R){ # trap

D[i,r,k] <- sqrt(pow(S[i,1,k]-X[r,1],2) + pow(S[i,2,k]-X[r,2],2)) # Squared distance to trap

g[i,k,r+1] <- exp(-pow(D[i,r,k]/sigma[gr[i]], kappa[gr[i]])) # Trap exposure

}

G[i,k] <- sum(g[i,k,]) # Total trap exposure

for(j in 1:J[i,k]){

P[i,j,k] <- (1 - exp(-lambda[tod[k,j],gr[i]]\*G[i,k]))\*z[i,k] # Probability of being captured

PI[i,k,j] <- step(H[i,j,k]-2)\*(g[i,k,H[i,j,k]]/(G[i,k] + 0.000000001))\*P[i,j,k] + (1-step(H[i,j,k]-2))\*(1-P[i,j,k])

Ones[i,j,k] ~ dbern(PI[i,k,j])

}

}

}

}

",fill = TRUE)

sink()

##########################################################################

### FUNCTION FOR SIMULATING CAPTURE HISTORY - Spatial robust design ###

### model with zero-inflated exponential dispersal distribution ###

##########################################################################

#

# The function allows two groups of individuals (e.g. males and females) and

# two types of secondary trapping sessions (e.g. morning and evening). The model

# is the same as the JAGS model specification above, with zero inflation. The

# function uses excessive looping for easier comparison (and checking) with the

# JAGS model specification).

#

# The function can be used for posterior predictive checks by iteratively

# sampling 'par' from the posterior distribution, generating new data (keeping

# the other function arguments (design variables) corresponding to the original

# data), and comparing aspects of the simulated data with the original data.

# ARGUMENTS:

# - par: List of parameters with the same definition as in the JAGS model

# specification above

# - K, J, first, gr, tod, S.first: Design variables with the same definition

# as in the JAGS model specification above.

# S.first[i,] is the first centre of activity

# of individual i

sim.exp.zeroinflated = function(par,K,J,first,gr,tod,S.first,X){

N = c(sum((K-first)==0), length(first))

R = nrow(X)

n.prim = max(K)

n.sec = max(J)

lambda = matrix(NA,2,2)

for(sex in 1:2){

for(TOD in 1:2){

lambda[TOD, sex] = exp(par$log.lambda0) \* par$beta[1]^(TOD-1) \* par$beta[2]^(sex-1)

}

}

S = array(NA, c(n,2,n.prim))

for(i in 1:N[2]){

S[i,,first[i]] = S.first[i,]

}

D = array(NA, c(n,n.prim,R))

g = array(NA, c(n,n.prim,R+1))

G = array(NA, c(n,n.prim))

P = array(NA, c(n,n.prim,max(J)))

PI = array(NA, c(n,n.prim,max(J),R))

z = array(NA,c(n,n.prim))

u = array(NA,c(n,n.prim-1))

Palive = array(NA, c(n,n.prim-1))

dd = d = theta = array(NA,c(n,n.prim-1))

H = array(NA, c(N[2],n.sec,n.prim))

# Loop over individuals that are only seen in the last primary session or the

# session they are censored

for(i in 1:N[1]){

z[i,first[i]] = 1

g[i,first[i],1] = 0

for(r in 1:R){ # trap

D[i,first[i],r] = sqrt((S[i,1,first[i]]-X[r,1])^2 + (S[i,2,first[i]]-X[r,2])^2)

g[i,first[i],r+1] = exp(-(D[i,first[i],r]/par$sigma[gr[i]])^par$kappa[gr[i]])

}

G[i,first[i]] = sum(g[i,first[i],])

while(!any(H[i,,first[i]]!=1, na.rm=T)){ # We condition on first primary session

for(j in 1:J[i,first[i]]){

P[i,first[i],j] = 1 - exp(-lambda[tod[first[i],j],gr[i]]\*G[i,first[i]]) # Probability of being captured

PI[i,first[i],j,] = (g[i,first[i],2:(R+1)]/(G[i,first[i]]+ 0.000000001))\*P[i,first[i],j]

H[i,j,first[i]] = sample(1:(R+1),1, prob = c(1-P[i,first[i],j],PI[i,first[i],j,]))

}

}

}

# Loop over all other individuals:

for(i in (N[1]+1):N[2]){ # ind

z[i,first[i]] = 1

## FIRST PRIMARY SESSION ##

g[i,first[i],1] = 0

for(r in 1:R){ # trap

D[i,first[i],r] = sqrt((S[i,1,first[i]]-X[r,1])^2 + (S[i,2,first[i]]-X[r,2])^2)

g[i,first[i],r+1] = exp(-(D[i,first[i],r]/par$sigma[gr[i]])^par$kappa[gr[i]])

}

G[i,first[i]] = sum(g[i,first[i],])

while(!any(H[i,,first[i]]!=1, na.rm=T)){ # We condition on first primary session

for(j in 1:J[i,first[i]]){

P[i,first[i],j] = 1 - exp(-lambda[tod[first[i],j],gr[i]]\*G[i,first[i]]) # Probability of being captured

PI[i,first[i],j,] = (g[i,first[i],2:(R+1)]/(G[i,first[i]]+ 0.000000001))\*P[i,first[i],j]

H[i,j,first[i]] = sample(1:(R+1),1, prob = c(1-P[i,first[i],j],PI[i,first[i],j,]))

}

}

## LATER PRIMARY SESSIONS ##

for(k in (first[i]+1):K[i]){ # primary session

theta[i,k-1] = runif(1,-3.141593,3.141593)

Palive[i,k-1] = z[i,k-1]\*par$phi[gr[i],k-1]

z[i,k] = rbinom(1,1,Palive[i,k-1])

u[i,k-1] = rbinom(1,1,par$psi[gr[i],k-1])

dd[i,k-1] = rexp(1, 1/par$dmean[gr[i]])

d[i,k-1] = u[i,k-1]\*(par$d.offset[gr[i]] + dd[i,k-1])

S[i,1,k] = S[i,1,k-1] + d[i,k-1]\*cos(theta[i,k-1])

S[i,2,k] = S[i,2,k-1] + d[i,k-1]\*sin(theta[i,k-1])

g[i,k,1] = 0

for(r in 1:R){ # trap

D[i,k,r] = sqrt((S[i,1,k]-X[r,1])^2 + (S[i,2,k]-X[r,2])^2)

g[i,k,r+1] = exp(-(D[i,k,r]/par$sigma[gr[i]])^par$kappa[gr[i]])

if(is.na(g[i,k,r+1])) g[i,k,r+1] = 0.0000000001

}

G[i,k] = sum(g[i,k,]) # Total trap exposure

for(j in 1:J[i,k]){

P[i,k,j] = (1 - exp(-lambda[tod[k,j],gr[i]]\*G[i,k]))\*z[i,k] # Probability of being captured

PI[i,k,j,] = (g[i,k,2:(R+1)]/(G[i,k]+ 0.000000001))\*P[i,k,j]

H[i,j,k] = sample(1:(R+1),1, prob = c(1-P[i,k,j],PI[i,k,j,]))

}

}

} # ind

H[is.na(H)] = 1

H

}

##############################################

### FUNCTION FOR GENERATING INITIAL VALUES ###

##############################################

# May need to be adjusted based on what are plausible starting values

Inits = function(H,X,K){

n = dim(H)[1]

n.prim = dim(H)[3]

mean.x = apply(H, c(1,3), function(i) mean(X[i-1,1], na.rm=T))

mean.y = apply(H, c(1,3), function(i) mean(X[i-1,2], na.rm=T))

# For initial values of dispersal distance:

for(i in (n.prim-1):1){

mean.x[,i][is.na(mean.x[,i])] = mean.x[,i+1][is.na(mean.x[,i])]

mean.y[,i][is.na(mean.y[,i])] = mean.y[,i+1][is.na(mean.y[,i])]

}

dx = mean.x[,2:n.prim] - mean.x[,1:(n.prim-1)]

dy = mean.y[,2:n.prim] - mean.y[,1:(n.prim-1)]

d.emp = sqrt(dx^2 + dy^2)

ch = apply(H,c(1,3), function(i) any(i!=1))

first.last = apply(ch, 1, function(i) range(which(i)))

z = ch

theta = atan2(dy,dx)

theta[is.na(theta)] = runif(sum(is.na(theta)), -pi, pi)

d = d.emp

d[is.na(d)] = 0.001

S = array(NA, c(n,2,n.prim)) # For initial values og FIRST location

for(i in 1:n){

S[i,1,first.last[1,i]] = mean.x[i,first.last[1,i]]

S[i,2,first.last[1,i]] = mean.y[i,first.last[1,i]]

z[i, first.last[1,i]:first.last[2,i]] = 1 # 1 when known to be alive, 0 otherwise

if(first.last[1,i] != 1){

theta[i,1:(first.last[1,i]-1)] = NA

z[i,1:(first.last[1,i]-1)] = NA

d[i,1:(first.last[1,i]-1)] = NA

}

if(K[i]!=n.prim){ # Adding NA's for censored individuals

theta[i,K[i]:(n.prim-1)] = NA

z[i, (K[i]+1):n.prim] = NA # after being removed

d[i, K[i]:(n.prim-1)] = NA

}

}

list(

kappa = runif(2, 1.9, 2.1),

sigma = runif(2, 4, 11), # Do not set too high if kappa is low

PL = runif(1,0.7,0.9),

beta = runif(2,0.5,2),

dmean = c(10,15) + runif(2,-3,3),

Phi = runif(2,0.5,0.8),

theta = theta,

d = d,

z = z,

S = S

)

}

###################################################################

### SETTING UP (STOCHASTIC) SAMPLING DESIGN AND SIMULATING DATA ###

###################################################################

## SETTING UP TRAPPING GRID: ##

# Setting up a 10x10 grid with spacing distance 'trap.dist' between traps

trap.dist = 7

X.max = Y.max = 10

X = expand.grid(

x = seq(trap.dist, X.max\*trap.dist, trap.dist),

y = seq(trap.dist, Y.max\*trap.dist, trap.dist)

)

## SETTING UP SESSIONS AND SAMPLE CHARACTERISTICS: ##

n.prim = 4 # number of primary sessions

n.sec = 5 # number of secondary sessions in each primary session

dt = runif(n.prim-1,0.8,1.2) # time between primary sessions

# Releasing 100 individuals in first primary session + 20 new in each primary thereafter

first = c(rep(1,100), rep(2:n.prim, each=20))

n = length(first)

gr = sample(1:2, n, replace=T)

K = rep(n.prim, n)

J = matrix(n.sec, n, n.prim)

# Letting every other secondary sesson be morning and evening:

tod = matrix(c(1,2), ncol=n.sec, nrow=n.prim, byrow=T)

# Random initial centres of activities

bord = 2\*trap.dist # How far outside the trapping grid can first centre og activity be located?

S.first = cbind(runif(n, -bord, X.max+bord), runif(n, -bord, Y.max+bord))

# Random censoring of 10 random individuals:

cens = sample(1:n, 10)

for(i in cens){

k = first[i]+sample(0:(n.prim-first[i]),1)

s = sample(1:n.sec,1)

K[i] = k

J[i,k] = s

}

# For the JAGS model, individuals must be sorted such that the individuals

# that are only seen in the last primary session or the session they are

# censored comes first:

ord = order(K-first)

K = K[ord]

J = J[ord,]

first = first[ord]

gr = gr[ord]

S.first = S.first[ord,]

## SETTING UP MODEL PARAMETERS: ##

# Here setting all psi to 1 and the d.offset to 0 to generate data from an

# exponential dispersal model without zeroinflation

Phi = c(0.7,0.9) # survival per time unit of males and females

par = list(

log.lambda0 = log(-log(1-0.8)),

beta = c(2,0.5),

kappa = c(1.5,2.5),

sigma = c(5,10),

psi = matrix(c(1, 1), nrow=2, ncol=length(dt)),

dmean = c(10,15),

d.offset = c(0,0),

phi = matrix(c(Phi[1]^dt, Phi[2]^dt), nrow=2, byrow=T)

)

##################################################################

### SIMULATING CAPTURE HISTORIES AND FITTING THE MODEL IN JAGS ###

##################################################################

# Simulating capture history data

H = sim.exp.zeroinflated(par,K,J,first,gr,tod,S.first,X)

# Priors for first centre of activity:

# Assuming that first centre of activity is always within +/- maxd

# from the mean capture location in both x and y direction.

maxd = 2\*trap.dist

mean.x = apply(H, c(1,3), function(i) mean(X[i-1,1], na.rm=T))

mean.y = apply(H, c(1,3), function(i) mean(X[i-1,2], na.rm=T))

first.mean.x = apply(mean.x,1, function(i) i[min(which(!is.na(i)))])

first.mean.y = apply(mean.y,1, function(i) i[min(which(!is.na(i)))])

xlow = first.mean.x - maxd

xupp = first.mean.x + maxd

ylow = first.mean.y - maxd

yupp = first.mean.y + maxd

JAGS.data = list(

N = c(sum((K-first)==0), length(first)),

K = K,

R = nrow(X),

J = J,

tod = tod,

first = first,

X = as.matrix(X),

H = H,

n.prim = dim(H)[3],

dt = dt,

gr = gr,

Ones = array(1, dim(H)),

# Prior parameters for initial centre of activity:

xlow = xlow,

xupp = xupp,

ylow = ylow,

yupp = yupp

)

library(rjags)

# Parameters to monitor

#params = c("kappa", "sigma", "log.lambda0", "beta", "Psi0", "psi", "dmean", "phi", "Phi", "d.offset")

params = c("kappa", "sigma", "log.lambda0", "beta", "dmean", "phi", "Phi")

# Generating initial values for 3 chains

inits = list(Inits(H,X=X,K=K), Inits(H,X=X,K=K),Inits(H,X=X,K=K))

# Fitting the model in JAGS (may take several hours, and may need to be run for

# for much longer to get good convergence)

(t1=Sys.time())

jags = jags.model(file="RD-SCR.Exp.txt", data=JAGS.data, inits=inits, n.chains=3, n.adapt=1000)

out = coda.samples(jags, variable.names=params, n.iter=3000,thin=1)

(t2=Sys.time())

(t2-t1)

# Looking at the chains:

plot(out, ask=T)

# Summary:

summary(out)

#############################################################################

### GENERATING NEW CAPTURE HISTORY DATA BASED ON THE SAME DESIGN BUT WITH ###

### PARAMETERS SAMPLED FROM THE POSTERIOR DISTRIBUTION (MAY BE USED FOR ###

### GENERATING POSTERIOR PREDICTIVE SAMPLES OF CHOSEN TEST STATISTICS): ###

#############################################################################

posterior = as.matrix(out)

psamp = posterior[sample(1:nrow(posterior),1),]

posterior.sample = list(

log.lambda0 = psamp["log.lambda0"],

beta = psamp[c("beta[1]","beta[2]")],

kappa = psamp[c("kappa[1]","kappa[2]")],

sigma = psamp[c("sigma[1]","sigma[2]")],

# psi = 1 - matrix(c(psamp["Psi0[1]"]^dt, psamp["Psi0[2]"]^dt), nrow=2, byrow=T),

psi = matrix(c(0, 0), nrow=2, ncol=length(dt)),

dmean = psamp[c("dmean[1]","dmean[2]")],

# d.offset = psamp[c("d.offset[1]", "d.offset[2]")],

d.offset = c(0,0),

phi = matrix(c(psamp["Phi[1]"]^dt, psamp["Phi[2]"]^dt), nrow=2, byrow=T)

)

H.new = sim.exp.zeroinflated(posterior.sample,K,J,first,gr,tod,S.first,X)