SEGUIMIENTO DE LA DIVERSIDAD BIOLÓGICA

Captura-recaptura espacialmente explícita (SCR)

José Jiménez García-Herrera (IREC-CSIC)

Universidad de Castilla-La Mancha

Los modelos de captura-recaptura espacialmente explícitas (SCR) son herramientas avanzadas que sirven para realizar inferencias sobre los tamaños poblacionales (número y densidad) y las distribuciones espaciales de las especies, y por ello pueden ser utilizadas para estudios de selección de hábitat, conectividad, co-ocurrencia, etc. Los modelos abiertos (OpenSCR) permiten estimar además parámetros demográficos (natalidad y mortalidad/supervivencia) que son también espacialmente explícitos. Los modelos SCR resultan herramientas imprescindibles para conocer el estado de conservación real de especies elusivas.

Hay dos grupos de aproximaciones a SCR: MLE, con software de extraordinaria calidad, de uso sencillo y salidas gráficas muy atractivas, como *secr* (Efford, 2020) y *oSCR* (Sutherland et al., 2018), y bayesianas (Royle et al., 2014), que pueden resultar más complejas de aprender, pero que son extraordinariamente versátiles y pueden ser modificadas para adaptarse a diferentes necesidades. Estas últimas son las que vamos a estudiar.Vamos a ejecutarlas usando `R` (R Core Team, 2020) y Nimble (De Valpine et al., 2017).

Repetid los análisis que se ejecutan aquí, cambiando cada uno de vosotros los valores de número de ocasiones de muestreo K y el valor de *rnd* que controla la aleatoriedad de la simulación. Pegad los resultados en un procesador de texto (valores que habeis seleccionado, resultados numéricos y gráficos) y enviádmelo por e-mail a: Jose.Jimenez@csic.es. Para ampliar conocimientos resulta adecuado el libro *Spatial Capture-Recapture* de Royle et al. (2017) https://www.sciencedirect.com/book/9780124059399/spatial-capture-recapture.

1. Simulación de datos

Vamos a simular datos en `R` y a ejecutar el modelo SCR. En este caso vamos a trabajar con detección/no detección,con una distribución binomial, aunque es muy sencillo cambiarlo y trabajar con conteos, usando una distribución de Poisson (podeis verlo en el código). El nivel de precisión que podemos obtener con estas estimas va a depender en buena parte del número de recapturas, que a su vez depende de la detectabilidad (λ_0), el número de ocasiones

de muestreo K y la relación entre la distancia entre trampas/cámaras y el movimiento de los animales (σ). Para generar los datos simulados vamos a cargar varios paquetes de R y un conjunto de funciones que he compilado y modificado para ser utilizadas aquí.

```
> source("SCR_functions.R")  # Funciones de interés
> library(scrbook)           # Atención, scrbook no está en CRAN
> library(spatstat)
> library(lattice)
> library(coda)
```

Datos a simular. Aquí debeis cambiar vosotros el valor de K. La detectabilidad, si queréis cambiarla, debéis modificar la función original *SimSCR0*.

```
> N <- 50  # Tamaño de población
> K <- 15  # Ocasiones de muestreo
> J <- 100 # Número de trampas
```

Vamos a generar nuestros datos. Los veremos gráficamente y exploraremos con *str*:

```
> data <- SimSCR0(N=N, K=K, array3d = TRUE, discard0=TRUE, rnd=2013)
> str(data)
```

```
List of 10
 $ Y       : int [1:38, 1:100, 1:15] 0 0 0 0 0 0 0 0 0 0 ...
 $ traplocs: int [1:100, 1:2] 1 1 1 1 1 1 1 1 1 1 ...
 $ xlim    : num [1:2] -0.25 11.25
 $ ylim    : num [1:2] -0.25 11.25
 $ N       : num 50
 $ p0      : num 0.1
 $ alpha1  : num 2
 $ sigma   : num 0.5
 $ K       : num 15
 $ S       : num [1:50, 1:2] 5.08 10.7 8.78 8.53 2.65 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "sx" "sy"
```

```
> S<-data$S    # Ubicación espacial de todos los animales generados
> y3d<-data$Y  # Matriz de captura
> (detections <- sum(y3d)) # Total de detecciones
```

```
[1] 94
```

```
> (nind <- dim(y3d)[1])    # individuos detectados
```

```
[1] 38
```

```
> X <- data$traplocs       # trampas (o cámaras-trampa)
> K <- data$K              # ocasiones de captura
> J <- nrow(X)             # total de cámaras
> M <- 150 # Aumentado de datos
> # Espacio de estados
> xlim <- data$xlim        # ubicaciones x que limitan el espacio de estados
> ylim <- data$ylim        # ubicaciones y que limitan el espacio de estados
> area <- diff(xlim)*diff(ylim)       # área afectada por el muestreo
> y <- apply(y3d,c(1,2),sum); sum(y)  # matriz de captura reducida sobre K
```

```
[1] 94
```

```
> # Aumentado de datos:
> yaug<-array(0,c(M,J))
> yaug[1:nind,]<-y[1:nind,]
```

Vamos a plotear las capturas y crear el spiderplot. En el spiderplot representamos las trampas o detectores (en este caso utilizamos cruces; en la versión standard usando *scrbook* son puntos). Con puntos en rojo, representamos los individuos de la poblacion, diferenciando con relleno los observados, y con solo contornos, los no observados. Los círculos en gris transparentes son las observaciones realizadas en cada camara trampa, y tienen un tamaño proprorcional al número de observaciones. Los puntos en violeta son promedio de las detecciones de cada individuo en el total de cámaras donde se ha detectado, y vienen a ser una aproximación naïve de los centros de actividad. Las líneas en negro representan el movimiento del animal entre diferentes trampas (que lo denominamos recapturas espaciales).

```
> captured<-apply(y3d,1,sum)
> captured[captured>1]<-1
> captured<-(1:nind)*captured
> # Preparamos para hacer la representación del número de capturas por trampa
> datn<-apply(y3d, c(2,3), sum)
> tot<-apply(datn, 1,sum)
```

```
> plot(X, xlim=xlim, ylim=ylim, pch="+", type="n", xlab="X", ylab="Y",
+      asp=TRUE)
> points(S, pch=1, col="red", cex=1.5) # capturados: puntos sólidos
> points(S[captured,], pch=16, col="red", cex=1.5)
> symbols(X, circles=tot/5, inches=F, bg="#00000022", fg=NULL, add=T)
> points(X, pch="+", cex=1)
> spiderplotJJ4(y3d, X, buffer=2, lwd=2)
```

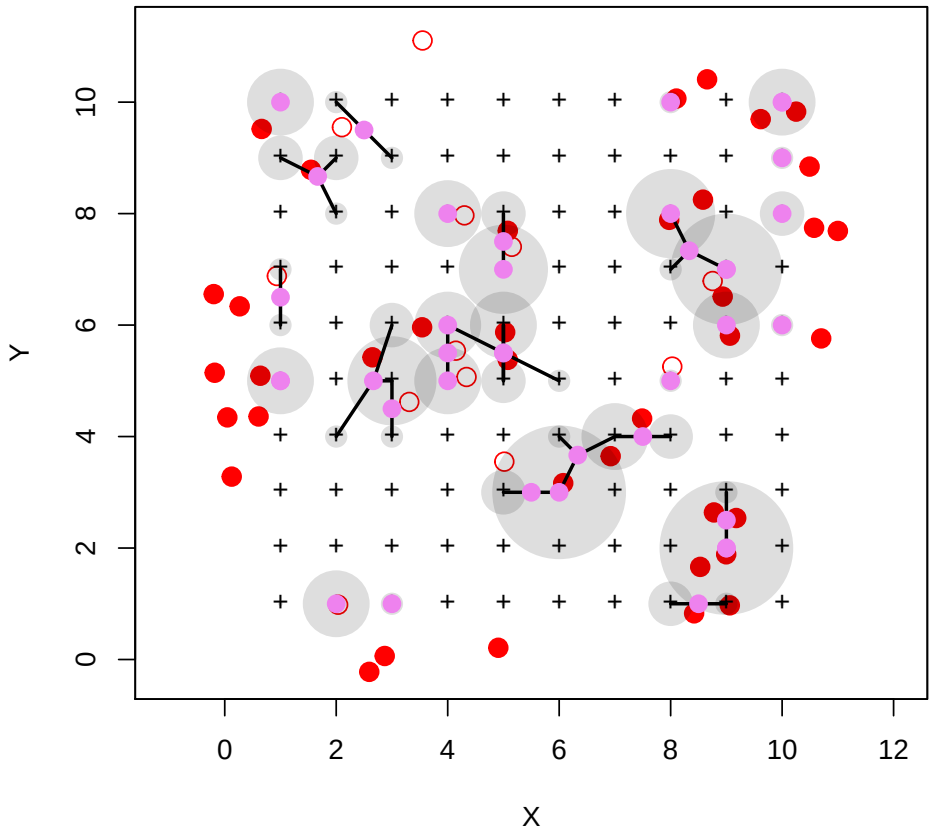


Figura 1: Spiderplot (ver exlicación en texto)

2. Modelo en BUGS usando Nimble

```
> library(nimble)
> code <- nimbleCode({  # definimos el modelo
+
+   alpha0 ~ dnorm(0,0.1)
+   logit(p0) <- alpha0
+   alpha1 ~ dnorm(0,0.1)
+   sigma <- sqrt(1/(2*alpha1))
+   psi ~ dunif(0,1)
+
+   for(i in 1:M){
+     z[i] ~ dbern(psi)
+     s[i,1] ~ dunif(xlim[1],xlim[2])
+     s[i,2] ~ dunif(ylim[1],ylim[2])
+     d2[i,1:J] <- pow(s[i,1]-X[1:J,1],2) + pow(s[i,2]-X[1:J,2],2)
+     p[i,1:J] <- p0*exp(-alpha1*d2[i,1:J])*z[i]
+
+     for(j in 1:J){
+       y[i,j] ~ dbinom(p[i,j],K)
+       #y[i,j] ~ dpois(p[i,j]*K) # Alternativamente podemos usar una
+                                 # distribución de Poisson para trabajar
+                                 # con conteos
+     }
+   }
+
+   N <- sum(z[1:M])
+   D <- N/area
+ })
>
```

Preparación de datos, constantes e inicios:

```
> # Inicios para las ubicaciones latentes
> sst <- cbind(runif(M,xlim[1],xlim[2]),runif(M,ylim[1],ylim[2]))
> for(i in 1:nind){
+   sst[i,1] <- mean( X[y[i,]>0,1] )
+   sst[i,2] <- mean( X[y[i,]>0,2] )
```

```
+ }
> str(constants <- list(M=M,          # aumentado de datos
+                       K=K,          # ocasiones de muestreo
+                       J=J,          # número de trampas
+                       area=area)) # área del espacio de estados

List of 4
 $ M   : num 150
 $ K   : num 15
 $ J   : int 100
 $ area: num 132

> str( dataN  <-   list(y=yaug,       # matriz de capturas reducida
+                       X=X,          # matriz de coordenadas de las trampas
+                       xlim=xlim,  # extremos x del espacio de estados
+                       ylim=ylim)) # extremos y del espacio de estados

List of 4
 $ y   : num [1:150, 1:100] 0 0 0 0 0 0 0 0 0 0 ...
 $ X   : int [1:100, 1:2] 1 1 1 1 1 1 1 1 1 1 ...
 $ xlim: num [1:2] -0.25 11.25
 $ ylim: num [1:2] -0.25 11.25

> str( inits   <-  list(p0=0.5,       # probabilidad basal de detección
+                       alpha1=1,     # parametrización de sigma
+                       s=sst,        # ubicaciones de inicio
+                       z=c(rep(1, nind), rbinom((M-nind),1,0.2))))

List of 4
 $ p0    : num 0.5
 $ alpha1: num 1
 $ s     : num [1:150, 1:2] 5 10 9 8.5 2.67 ...
 $ z     : num [1:150] 1 1 1 1 1 1 1 1 1 1 ...

>
```

3. Compilación y ejecución

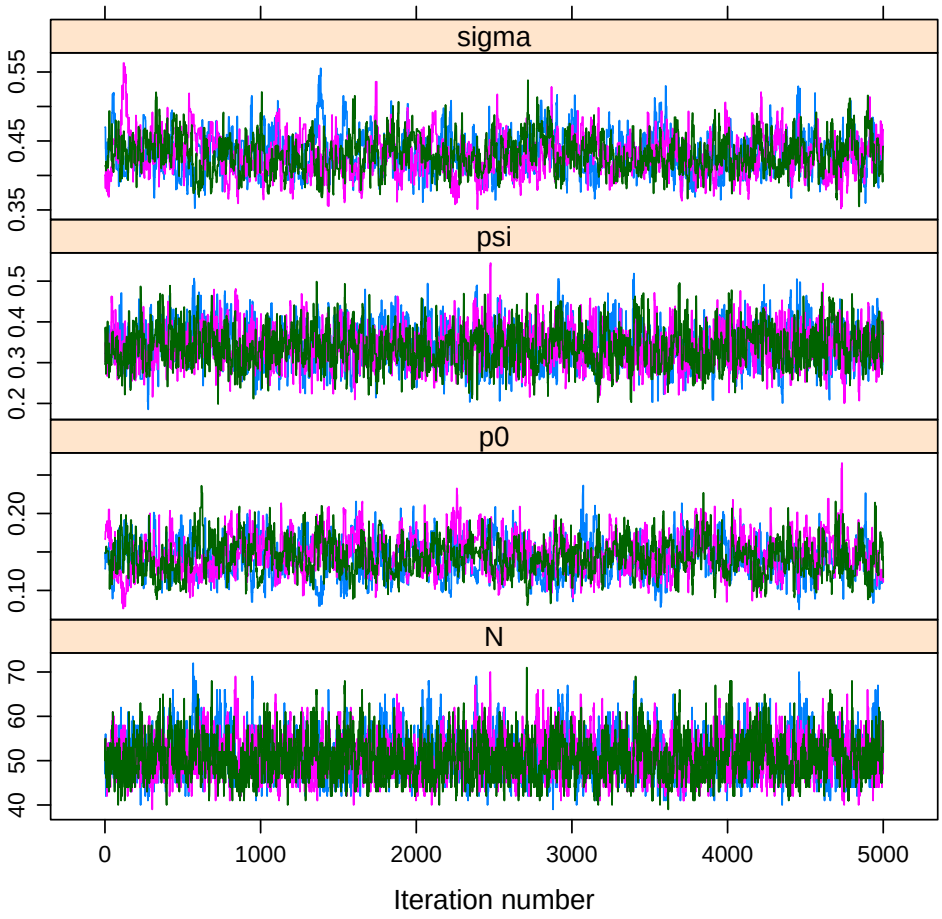
```
> # Preparamos el modelo para ejecución en Nimble
> Rmodel <- nimbleModel(code=code,
```

```
+                       constants=constants,
+                       data=dataN,
+                       inits=inits,
+                       check=FALSE,
+                       calculate=FALSE)
> Cmodel <- compileNimble(Rmodel)
> # Establecemos los parámetros a monitorizar
> params<-c('N', 'D', 'sigma','psi', 'p0','s','z')
> mcmcspec<-configureMCMC(Rmodel, monitors=params)

===== Monitors =====
thin = 1: N, D, sigma, psi, p0, s, z
===== Samplers =====
RW sampler (303)
  - alpha0
  - alpha1
  - psi
  - s[]  (300 elements)
binary sampler (150)
  - z[]  (150 elements)

> # Cambiamos el muestreador de z (opcional)
> mcmcspec$removeSamplers('z')
> for(node in Rmodel$expandNodeNames('z')) mcmcspec$addSampler(target = node,
+                                                              type = 'slice')
> mcmcspec$removeSamplers("s")
> ACnodes <- paste0("s[", 1:constants$M, ", 1:2]")
> for(node in ACnodes) {
+     mcmcspec$addSampler(target = node,
+                         type = "RW_block",
+                         control = list(adaptScaleOnly = TRUE),
+                         silent = TRUE)
+ }
> # Construimos el modelo
> scrMCMC <- buildMCMC(mcmcspec)
> # Compilamos
> CSCRMCMC <- compileNimble(scrMCMC, project = Rmodel)
> # Ejecutamos el modelo
```

```
> nb=1000      # Iteraciones a desechar
> ni=5000 +nb  # Iteraciones
> nc=3         # Cadenas
> start.time2<-Sys.time()
> outNim <- runMCMC(CSCRMCMC,
+                   niter = ni,
+                   nburnin = nb,
+                   nchains = nc,
+                   inits=inits,
+                   setSeed = TRUE,
+                   progressBar = TRUE,
+                   samplesAsCodaMCMC = TRUE)
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|

> end.time<-Sys.time()
> end.time-start.time2 # tiempo de ejecución

Time difference of 4.65594 mins
```

4. Resultados

```
> summary(outNim[,c('N','D','p0','psi', 'sigma')])

Iterations = 1:5000
Thinning interval = 1
Number of chains = 3
Sample size per chain = 5000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean      SD  Naive SE Time-series SE
N     50.7938 4.57948 0.0373913      0.1154936
D      0.3841 0.03463 0.0002827      0.0008733
p0     0.1436 0.02382 0.0001945      0.0008703
psi    0.3411 0.04872 0.0003978      0.0011643
sigma  0.4307 0.02810 0.0002294      0.0010936

2. Quantiles for each variable:

          2.5%     25%     50%     75%   97.5%
N     43.0000 48.0000 50.0000 54.0000 61.0000
D      0.3251  0.3629  0.3781  0.4083  0.4612
p0     0.1005  0.1269  0.1423  0.1593  0.1926
psi    0.2515  0.3065  0.3394  0.3731  0.4396
sigma  0.3805  0.4112  0.4284  0.4484  0.4905

> # Inspeccionamos la convergencia de las cadenas de Markov
> xyplot(outNim[,c('N','p0','psi', 'sigma')])
```

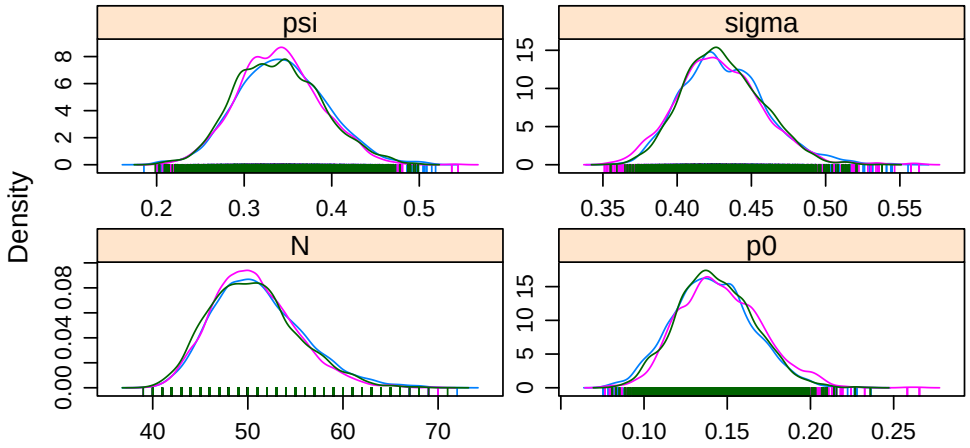



```
> gelman.diag(outNim[,c('N','p0','psi', 'sigma')], multivariate = FALSE)
Potential scale reduction factors:

      Point est. Upper C.I.
N           1.00       1.00
p0          1.01       1.04
psi         1.00       1.00
sigma       1.00       1.01
> cat("Población (N) que simulamos = ", data$N, "individuos", "\n")
```

```
Población (N) que simulamos =  50 individuos
> cat("p0 simulada  = ", data$p0, "\n")
p0 simulada  =  0.1
> cat("sigma simulada  = ", data$sigma, "\n")
sigma simulada  =  0.5
> cat("Datos usados = ", sum(yaug), "foto-capturas con identificación", "\n")
Datos usados =  94 foto-capturas con identificación
> samplesn<-data.matrix(outNim)
> # Coeficiente variación para N
> sd(samplesn[,2])/mean(samplesn[,2])
[1] 0.0901582
```

Para observar las distribuciones creadas

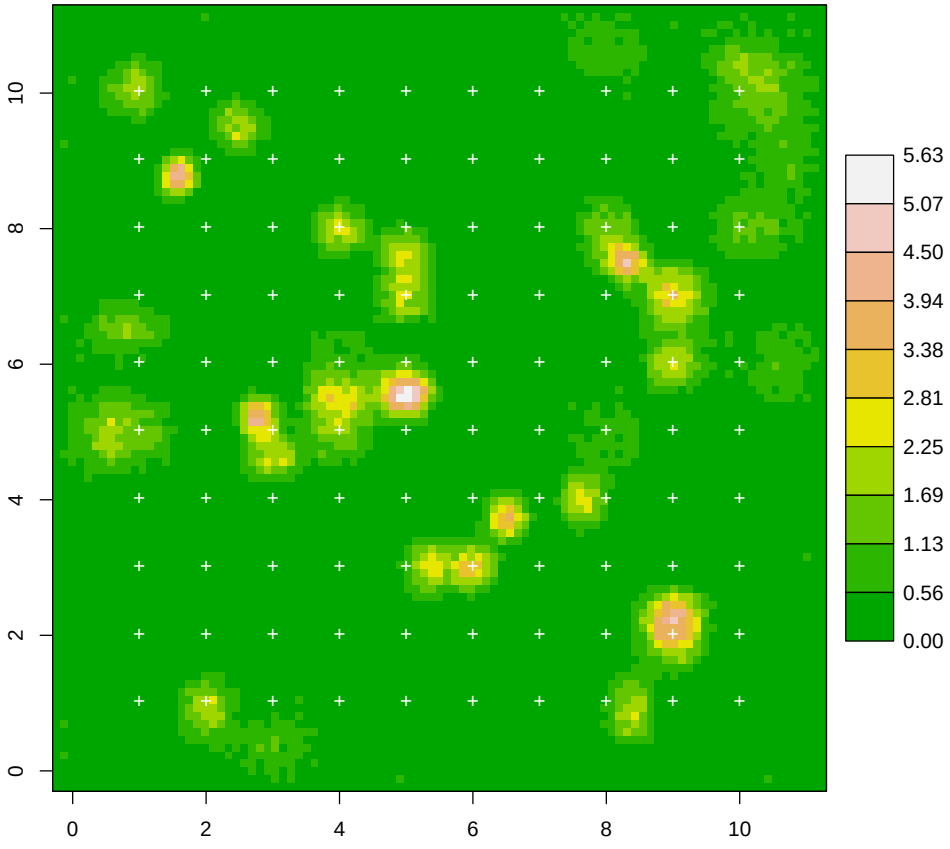
```
> densityplot(outNim[,c('N','p0','psi', 'sigma')])
```



El resultado gráfico es un ráster de probabilidades de la ubicación de los animales en el espacio:

```
> s1 <- samplesn[,c(5:154)]   ; Sx<-as.matrix(s1)
> s2 <- samplesn[,c(155:304)] ; Sy<-as.matrix(s2)
> z <-  samplesn[,c(306:455)] ; z<- as.matrix(z)
> delta<-1.3
> Xl<-min(X[,1])-delta
> Xu<-max(X[,1])+delta
> Yl<-min(X[,2])-delta
> Yu<-max(X[,2])+delta
```

```
> obj<-list(Sx=Sx,Sy=Sy,z=z)
> par(mfrow=c(1,1))
> Spat<-SCRdensity(obj, nx=100, ny=100, Xl=Xl, Xu=Xu, Yl=Yl, Yu=Yu)
mean:  0.3774807
> points(X, pch="+", col="white")
```



5. REFERENCIAS

De Valpine, P., Turek, D., Paciorek, C. J., Anderson-Bergman, D., Lang, T., & Bodik, R. (2017). Programming with models: writing statistical algorithms for general model structures with NIMBLE. *Journal of Computational and Graphical Statistics*, 26, 403–413.DOI:10.1080/10618600.2016.1172487

Efford, M. G. (2020). secr: Spatially explicit capture-recapture models. R package version 4.3.1. Retrieved from https://cran.r-project.org/package=secr

R Core Team. (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing. Vienna, Austria. Retrieved from https://www.r-project.org/.

Royle, J. A., Chandler, R. B., Sollmann, R., & Gardner, B. (2014). *Spatial capture-recapture*. Waltham, Massachusetts: Elsevier, Academic Press. doi:10.1016/B978-0-12-405939-9.00026-8

Sutherland, C. S., Royle, J. A., & Linden, D. W. (2018). oSCR: Multi-Session Sex-Structured Spatial Capture- Recapture Models. R package version 0.42.0.