

SEGUIMIENTO DE LA DIVERSIDAD BIOLÓGICA


Muestras de distancia jerárquicos (HDS)

José Jiménez García-Herrera (IREC-CSIC)

Universidad de Castilla-La Mancha

Este documento es una guía de un script que trabajaremos en clase. Repetid primero los análisis que se ejecutan aquí para asegurarnos de entender todo. Para practicar, cambiad cada uno de vosotros la densidad en la simulación.

Información sobre códigos

Los códigos de la primera parte (MLE) son del manual de ayuda online de **unmarked** en , e incluye desarrollos posteriores de Richard Chandler y Ken Kellner, que se pueden consultar en el grupo web de **unmarked** (<https://groups.google.com/g/unmarked>). Los de la segunda parte (bayesiano) son del libro de Kéry & Royle *Applied Hierarchical Modeling in Ecology Analysis of distribution, abundance and species richness in R and BUGS* (2016). Editorial Elsevier. Estos códigos se han modificado y adaptado para la docencia en este máster.

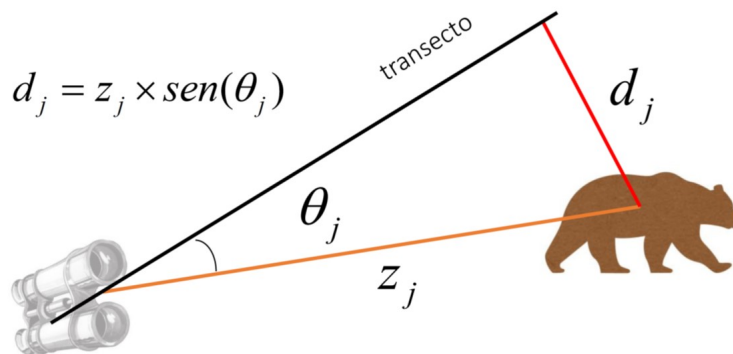


Figura 1: Muestreo de distancias

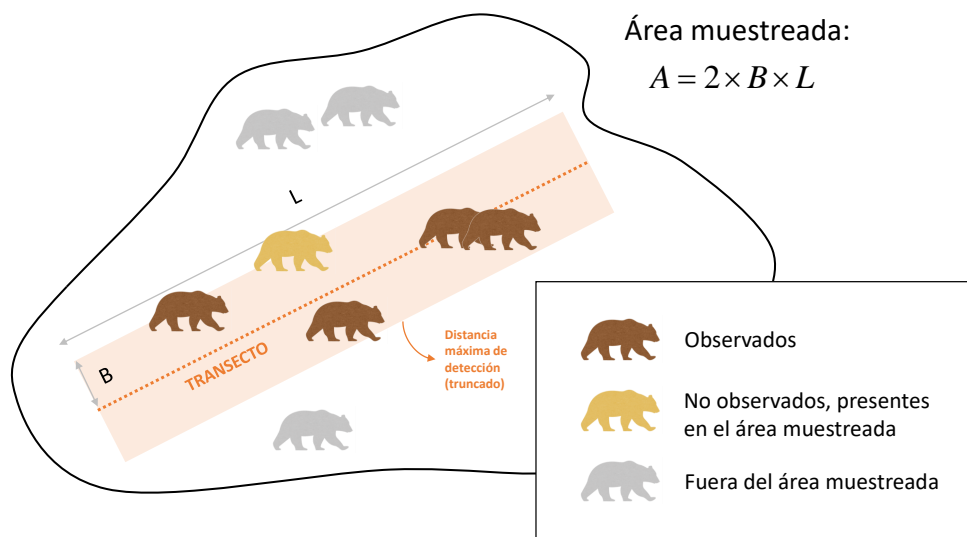


Figura 2: La detección de los animales -salvo para la función uniforme- es una función de la distancia desde el punto donde se encuentra el animal al eje del transecto.

Muestreos de Distancia Jerárquicos (HDS)

Los muestreos de distancias (Burnham et al., 1980) y por extensión los HDS son, a día de hoy, una de las familias de métodos más simples, económicos y fiables para la estima de poblaciones considerando la detectabilidad imperfecta. Los HDS permiten obtener simultáneamente estimas de densidad y detectabilidad, y usar covariables para ambas.

Premisas del muestreo de distancias

Hay dos tipos de muestreo de distancias: de líneas y de puntos. Sólo trataremos aquí el primero, pero ambos tienen la misma base teórica. En los muestreos de distancias de líneas se muestrea el espacio con recorridos lineales, anotando las distancias de los animales objetivo a la línea de transecto (Figura 1). Salvo para la distribución uniforme, se asume que la probabilidad de detección disminuye con la distancia entre el animal y el eje del transecto (Figura 2). Esa disminución de la detección va a ser una función seminormal, *hazard rate*,

exponencial o uniforme.

1. Los individuos en la línea de observación son detectados con probabilidad $p=1$.
2. Los animales deben ser detectados (y medida su distancia antes de haya algún movimiento de reacción frente al observador).
3. Cada detección debe ser independiente. Si los animales están agrupados esta premisa no se cumple, y puede ser conveniente utilizar un código específico para ese caso.
4. No debe existir error de medición.

1. Simulación de datos

Como en casos anteriores vamos a simular los datos, de manera que creamos una población de parámetros conocidos, y vemos si la ejecución del código nos devuelve la población de partida. En este caso vamos a emplear dos códigos diferentes para los mismos datos que simulamos. El primero es en MLE usando `unmarked` (Fiske & Chandler, 2011), y el segundo es un código bayesiano usando `Nimble` (De Valpine et al., 2017).

```
> set.seed(1975)
> dist.sim<-function(Dha=1,
+                    sigma=250, # Parámetro de la seminormal
+                    wmax=500,  # anchura de banda muestreada
+                    L=1000,    # longitud de cada transecto
+                    nL=10){    # número de transectos
+   Dm2<-Dha/10000
+   lambda<-Dm2*L*wmax*2
+   n<-rpois(nL,lambda)
+   # Función de detección seminormal
+   p.detect<-function(x,sigma){
+     det.p<-exp(-(x^2/(2*sigma^2)))
+   }
+   dist.data<-data.frame(line=numeric(0),length=numeric(0),distance=numeric(0))
+   for(i in 1:nL) {
+     all.dist<-runif(n[i],0,wmax)
+     detections<-rbinom(length(all.dist),1,p.detect(all.dist,sigma=sigma))
+     obs.dist<-all.dist[detections==1]
+     line<-rep(i,length(obs.dist))
+   }
```

```
+ length<-rep(L,length(obs.dist))
+ new.data<-data.frame(line=line,length=L,distance=obs.dist)
+ dist.data<-rbind(dist.data,new.data)
+ }
+ return<-list(dist.data,n)
+ }
```

Vamos a seleccionar una densidad determinada (0.15 individuos/Ha); un valor de sigma (250) para la detectabilidad (parámetro de la distribución seminormal, que nos informa de como decrece la detectabilidad con la distancia), una distancia máxima de detección (500 metros) que es el ancho de banda a cada lado del transecto, la longitud de los transectos a simular (todos serán de 1000 metros e igual longitud, para simplificar) y el número de transectos a realizar (10).

```
> sim.data<-dist.sim(Dha=0.15, sigma=250, wmax=500, L=1000, nL=10)
> # Esto equivale a S=500*2*1000*10 (1000 ha; 0.15*1000=150 animales)
> dat<-sim.data[[1]]
> str(sim.data)
```

List of 2

```
$ : 'data.frame':      85 obs. of  3 variables:
  ..$ line      : int [1:85] 1 1 1 1 1 1 1 1 2 2 ...
  ..$ length    : num [1:85] 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 ...
  ..$ distance: num [1:85] 368.29 132.05 30.46 39.36 2.65 ...
$ : int [1:10] 15 11 16 17 12 15 12 15 8 14
```

```
> # Los datos creados son:
> head(sim.data[[1]])
```

	line	length	distance
1	1	1000	368.292987
2	1	1000	132.047409
3	1	1000	30.464540
4	1	1000	39.363790
5	1	1000	2.647477
6	1	1000	496.036192

Vamos a prepararlos para su tratamiento con `unmarked`:

```
> breaks<-seq(0,500,50)
> n<-length(breaks)-1
```

```
> labels<-as.numeric(1:n)
> # Agrupamos los datos de distancia por intervalos y calculamos
> # las frecuencias para cada intervalo de distancia
> sim.data[[1]]$binned.x<-cut(sim.data[[1]]$distance,
+                             breaks=breaks,labels=labels)
> y<-with(sim.data[[1]],table(line,binned.x))
> # Vamos a verlo:
> str(y)

'table' int [1:10, 1:10] 3 0 1 2 1 2 0 1 2 2 ...
- attr(*, "dimnames")=List of 2
 ..$ line      : chr [1:10] "1" "2" "3" "4" ...
 ..$ binned.x: chr [1:10] "1" "2" "3" "4" ...
```

Convertimos en matriz:

```
> class(y) <- "matrix"
```

Vamos a ver la matriz

```
> y

      binned.x
line 1 2 3 4 5 6 7 8 9 10
  1  3 0 3 0 0 0 0 1 0  1
  2  0 1 0 2 0 1 0 1 0  1
  3  1 3 1 2 3 0 0 0 0  0
  4  2 2 1 1 0 1 1 0 0  0
  5  1 2 1 2 2 2 0 0 0  0
  6  2 3 0 2 2 0 0 0 2  0
  7  0 3 1 2 1 0 0 0 0  1
  8  1 2 2 0 0 1 1 1 2  0
  9  2 2 0 0 1 0 0 0 0  0
 10  2 3 0 0 3 0 1 0 0  0
```

Y vamos a preparar las longitudes de los transectos:

```
> tlength<-with(sim.data[[1]],aggregate(length~line,FUN="mean"))$length
> tlength

[1] 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
```

2. Estima MLE con unmarked

Preparamos los datos para tratarlos en unmarked:

```
> library(unmarked)
> sim.data.input<-unmarkedFrameDS(y=y,
+                                dist.breaks=breaks,
+                                tlength=tlength,
+                                survey="line",
+                                # Aqui no tenemos covariables:
+                                siteCovs=NULL,
+                                unitsIn="m")
```

Veamos el data.frame que hemos creado:

```
> head(sim.data.input)
```

Data frame representation of unmarkedFrame object.

	y.1	y.2	y.3	y.4	y.5	y.6	y.7	y.8	y.9	y.10
1	3	0	3	0	0	0	0	1	0	1
2	0	1	0	2	0	1	0	1	0	1
3	1	3	1	2	3	0	0	0	0	0
4	2	2	1	1	0	1	1	0	0	0
5	1	2	1	2	2	2	0	0	0	0
6	2	3	0	2	2	0	0	0	2	0
7	0	3	1	2	1	0	0	0	0	1
8	1	2	2	0	0	1	1	1	2	0
9	2	2	0	0	1	0	0	0	0	0
10	2	3	0	0	3	0	1	0	0	0

Ahora, en primer lugar vamos a elegir la función de detección utilizando la selección de modelos de unmarked basado el criterio de información de Akaike. Aquí, como hemos simulado con una seminormal, lógicamente nos va seleccionar la seminormal.

```
> # Ajuste de los modelos competidores
> mod1<-distsamp(~1 ~1, data=sim.data.input, keyfun="halfnorm",
+               output="density", unitsOut="ha")
> mod2<-distsamp(~1 ~1, data=sim.data.input, keyfun="hazard",
+               output="density", unitsOut="ha")
> mod3<-distsamp(~1 ~1, data=sim.data.input, keyfun="exp",
```

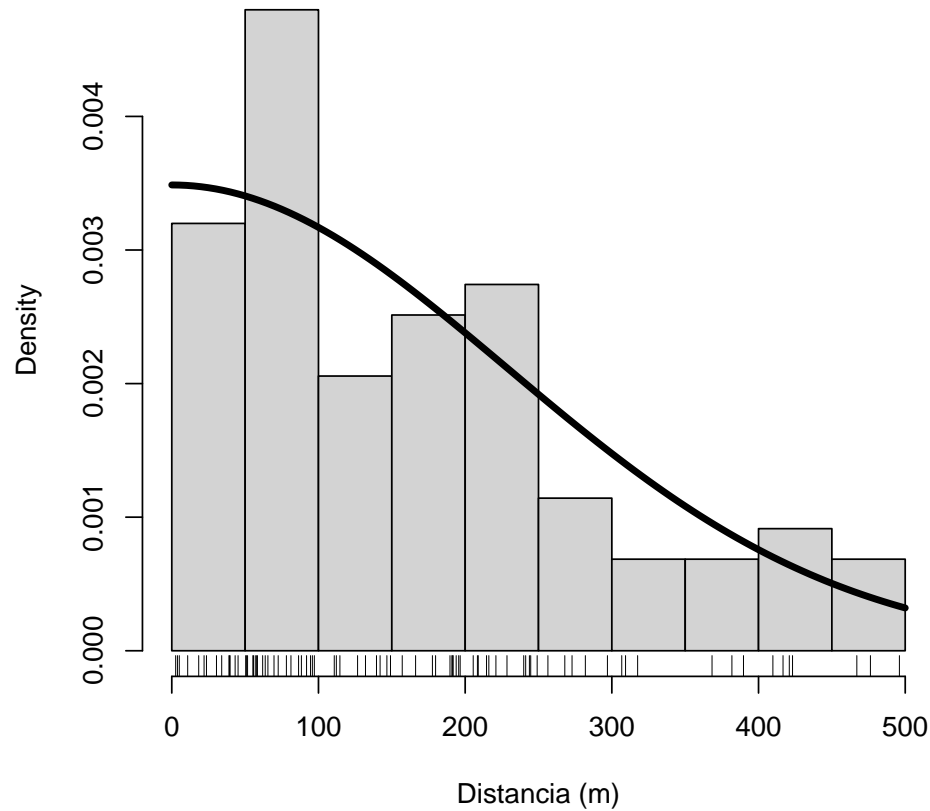
```
+               output="density", unitsOut="ha")
> mod4<-distsamp(~1 ~1, data=sim.data.input, keyfun="uniform",
+               output="density", unitsOut="ha")
> # Seleccionamos el modelo de menor AIC
> fmList <- fitList(Halfnormal=mod1, Hazard=mod2,
+               Exponential=mod3, Uniform=mod4)
> modSel(fmList)
```

	nPars	AIC	delta	AICwt	cumltvWt
Halfnormal	2	228.41	0.00	5.3e-01	0.53
Hazard	3	228.64	0.23	4.7e-01	1.00
Exponential	2	256.64	28.22	3.9e-07	1.00
Uniform	1	256.64	28.22	3.9e-07	1.00

El mejor modelo es -lógicamente- Halfnormal.

Veamos el histograma:

```
> hist(mod1, xlab="Distancia (m)", main="", col="lightgrey", lwd=4)
> x<-sim.data[[1]]$distance
> rug(x,side=1)
```



Inspeccionamos el modelo

```
> mod1
```

Call:

```
distsamp(formula = ~1 ~ 1, data = sim.data.input, keyfun = "halfnorm",
  output = "density", unitsOut = "ha")
```

Density:

Estimate	SE	z	P(> z)
-1.88	0.141	-13.3	1.71e-40

Detection:

Estimate	SE	z	P(> z)
5.43	0.108	50.3	0

AIC: 228.4134

En escala real:

```
> backTransform(mod1, type="state") # animales / ha
```

Backtransformed linear combination(s) of Density estimate(s)

Estimate	SE	LinComb	(Intercept)
0.153	0.0215	-1.88	1

Transformation: exp

```
> backTransform(mod1, type="det")
```

Backtransformed linear combination(s) of Detection estimate(s)

Estimate	SE	LinComb	(Intercept)
229	24.7	5.43	1

Transformation: exp

Vamos ahora a testar la bondad del ajuste:

```
> fitstats <- function(fm) {  
+   observed <- getY(fm@data)  
+   expected <- fitted(fm)  
+   resids <- residuals(fm)  
+   sse <- sum(resids^2)  
+   chisq <- sum((observed - expected)^2 / expected)  
+   freeTuke <- sum((sqrt(observed) - sqrt(expected))^2)  
+   out <- c(SSE=sse, Chisq=chisq, freemanTukey=freeTuke)  
+   return(out)  
+ }  
> (pb <- parboot(mod1, fitstats, nsim=1000, report=1))
```

```
Call: parboot(object = mod1, statistic = fitstats, nsim = 1000, report = 1)
```

```
Parametric Bootstrap Statistics:
```

	t0	mean(t0 - t_B)	StdDev(t0 - t_B)	Pr(t_B > t0)
SSE	74.4	-9.35	16.72	0.701
Chisq	99.8	1.57	14.08	0.413
freemanTukey	40.7	2.29	4.17	0.281

```
t_B quantiles:
```

	0%	2.5%	25%	50%	75%	97.5%	100%
SSE	40	55	72	83	93	122	194
Chisq	61	73	89	97	106	127	167
freemanTukey	26	30	36	38	41	47	50

```
t0 = Original statistic computed from data
```

```
t_B = Vector of bootstrap samples
```

```
> # chisquare
```

```
> (c.hat <- pb@t0[2] / mean(pb@t.star[,2])) # c-hat: ratio observado/esperado
```

```
Chisq
```

```
1.016012
```

Lo representamos gráficamente. Cuanto mejor coincida la línea de puntos con la mediana de la distribución normal que se representa en cada caso, mejor es la bondad del ajuste.

```
> par(mfrow=c(2,2))
```

```
> hist(pb@t.star[,1], xlab='SSE', col="grey90", breaks=15, main="")
```

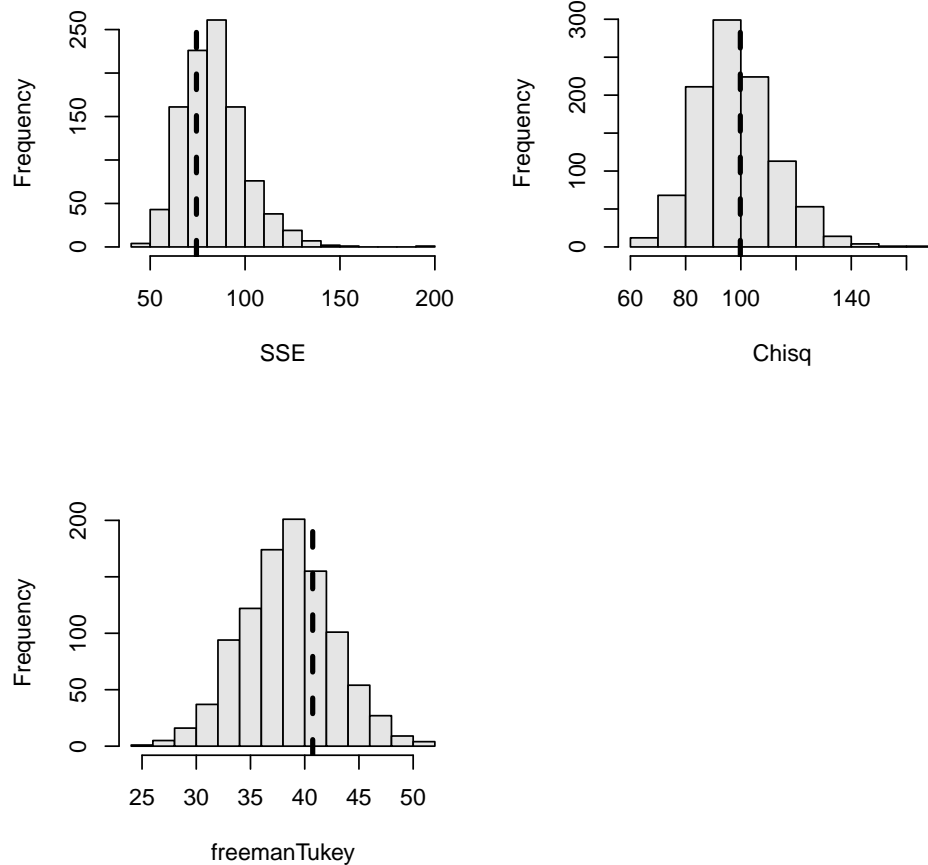
```
> abline(v=pb@t0[1],lty=2, lwd=3)
```

```
> hist(pb@t.star[,2], xlab='Chisq', col="grey90", breaks=15, main="")
```

```
> abline(v=pb@t0[2],lty=2, lwd=3)
```

```
> hist(pb@t.star[,3], xlab='freemanTukey', col="grey90", breaks=15, main="")
```

```
> abline(v=pb@t0[3],lty=2, lwd=3)
```



Habitualmente se reseña el chisq. como medida de la bondad del ajuste. Cuanto más cerca de 1 está, mejor es el ajuste.

```
> (c.hat <- pb@t0[2] / mean(pb@t.star[,2]))
```

```
Chisq
1.016012
```

Vamos a estimar la abundancia en cada transecto utilizando **ranef**, una aplicación bayesiana incluida en **unmarked**, conjuntamente con BUP (*Best Unbiased Predictors*), y vamos a compararlo con los datos originales que habíamos creado:

```
> re <- ranef(mod1, K=50)
> # Best Unbiased Predictors
> media<-bup(re, stat="mean", real)
> ci<-confint(re, level=0.9) # 90% CI
> (comp<-data.frame('media'=media, 'lower'=ci[,1],
+                   'upper'=ci[,2], 'real'=sim.data[[2]]))

      media lower upper real
1  14.76074    11    19    15
2  12.76074     9    17    11
3  16.76074    13    21    16
4  14.76074    11    19    17
5  16.76074    13    21    12
6  17.76074    14    22    15
7  14.76074    11    19    12
8  16.76074    13    21    15
9  11.76074     8    16     8
10 15.76074    12    20    14

> # Real
> cat("Poblacion simulada = ", sum(comp[,4]), "individuos", "\n")

Poblacion simulada = 135 individuos

> # Estimado
> cat("Poblacion estimada = ", round(sum(comp[,1]),0),
+     "(",sum(comp[,2]),"-",sum(comp[,3]),") individuos\n")


Poblacion estimada = 153 ( 115 - 195 ) individuos
```

Para obtener la desviación estándar y los intervalos de confianza empleamos esta función auxiliar que nos permite obtener la predicción posterior de la abundancia:

```
> # Función auxiliar
> postPredN <- function(ranefOut, nSims=100) {
+   post <- ranefOut$post
+   postDim <- dim(post)
+   nSites <- postDim[1]
+   nProbs <- postDim[2]
+   possibleN <- 0:(nProbs-1)
+   nYears <- postDim[3]
```

```
+   NsitePostSamples <- array(NA_integer_,
+                             c(nSites, nYears, nSims))
+
+   for(i in 1:nSites) {
+     for(t in 1:nYears) {
+       NsitePostSamples[i,t,] <- sample(
+         possibleN, size=nSims,
+         replace=TRUE, prob=prob[i,,t])
+     }
+   }
+   NpostSamples <- apply(NsitePostSamples, c(2,3), sum)
+   return(NpostSamples)
+ }
> # Obtenemos media, mediana, SD y CI95%
> Npost <- postPredN(re, nSims=1000)
> rowMeans(Npost)          ## media
[1] 152.921
> apply(Npost, 1, median)   ## mediana
[1] 153
> apply(Npost, 1, sd)       ## SD
[1] 8.265778
> apply(Npost, 1, quantile,
+       prob=c(0.025, 0.975)) ## 95% CI
      [,1]
2.5%    136
97.5%   169
```

3. Estima Bayesiana con NIMBLE

Vamos a usar los mismos datos simulados que hemos tratado en unmarked para ajustar un modelo HDS pero usando una aproximación bayesiana con Nimble (De Valpine et al., 2017) en  (R Core Team, 2020).

Preparamos los datos:

```
> # Vamos a escalar los datos de distancia
> dat$distance<-dat$distance/100
> dat$y<-1
> dat<-data.frame(site=dat$line,y=dat$y,d=dat$distance)
> B<-5
> # Vamos a obtener el número de individuos detectados por sitio
> # ncap = 1 más el número de individuos detectados por sitio
> ncap <- table(dat[,1])          # ncap = 1 si no hay observaciones
> sites0 <- dat[is.na(dat[,2]),][,1] # sitios sin detecciones
> ncap[as.character(sites0)] <- 0  # Rellenamos con 0 los sitios sin
>                                # detecciones
> ncap <- as.vector(ncap)
> # Preparamos los datos
> site <- dat[!is.na(dat[,2]),1]   # ID de sitio por observacion
> delta <- 0.1                    # agrupamos distancias en franjas
> midpt <- seq(delta/2, B, delta) # puntos medios de las franjas
> dclass <- dat[,3] %/% delta + 1  # convertimos distancias en grupos
> nD <- length(midpt)             # Número de intervalos de distancia
> dclass <- dclass[!is.na(dat[,2])] # Categorías observadas
> nind <- length(dclass)          # Número total de individuos observados
```

Preparamos el modelo:

```
> library(nimble)
> ## define the model
> code <- nimbleCode({
+   # Priors
+   alpha0 ~ dunif(-10,10)
+   beta0 ~ dunif(-10,10)
+
+   for(i in 1:nind){
```

```
+   dclass[i] ~ dcat(fc[site[i],1:nD]) # Parte 1 del HDS
+ }
+
+ for(s in 1:nsites){
+   # Construimos las probabilidades por celdas
+   for(g in 1:nD){           # midpt = punto central de cada celda
+     log(p[s,g]) <- -midpt[g] * midpt[g] / (2*sigma[s]*sigma[s])
+     pi[s,g] <- delta / B     # probabilidad por intervalo
+     f[s,g] <- p[s,g] * pi[s,g]
+     fc[s,g] <- f[s,g] / pcap[s]
+   }
+   pcap[s] <- sum(f[s,1:nD])      # Pr(captura): suma de las areas
+
+   ncap[s] ~ dbin(pcap[s], N[s])  # Parte 2 del HDS
+   N[s] ~ dpois(lambda[s])        # Parte 3 del HDS
+   log(lambda[s]) <- beta0
+   log(sigma[s]) <- alpha0
+ }
+ # Parámetros derivados
+ Ntotal <- sum(N[1:nsites])
+ area<- nsites*1*2*B # Unidad long.== 1, franja = 2xB
+ D<- Ntotal/(10*area) # animales por 100 ha
+ })
```

Suministramos ahora datos, constantes e inicios

```
> str(data <- list(midpt=midpt, ncap=ncap, dclass=dclass))
```

List of 3

```
$ midpt : num [1:50] 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95 ...
$ ncap  : num [1:10] 8 6 10 8 10 11 8 10 5 9
$ dclass: num [1:85] 37 14 4 4 1 50 15 15 18 48 ...
```

```
> str(constants<-list(nsites=10, site=site, nind=nind, B=B, nD=nD, delta=delta))
```

List of 6

```
$ nsites: num 10
$ site  : int [1:85] 1 1 1 1 1 1 1 1 2 2 ...
$ nind  : int 85
$ B     : num 5
```

```
$ nD      : int 50
$ delta   : num 0.1

> Nst <- ncap + 1
> str(inits <- list(alpha0=0, beta0=0, N=Nst))

List of 3
 $ alpha0: num 0
 $ beta0  : num 0
 $ N      : num [1:10] 9 7 11 9 11 12 9 11 6 10

Especificamos los parámetros a guardar

> params <- c('alpha0', 'beta0', 'Ntotal', 'D')

Compilamos y ejecutamos el modelo

> # Preparamos el modelo para ejecución en Nimble
> Rmodel <- nimbleModel(code=code, constants=constants, data=data,
+                       inits=inits, check=FALSE)
> Cmodel <- compileNimble(Rmodel)
> # Establecemos los parámetros a monitorizar
> mcmcspec<-configureMCMC(Rmodel, monitors=params, nthin=5)

===== Monitors =====
thin = 1: alpha0, beta0, D, Ntotal
===== Samplers =====
slice sampler (10)
- N[] (10 elements)
RW sampler (2)
- alpha0
- beta0

> # Construimos el modelo
> hdsMCMC <- buildMCMC(mcmcspec)
> # Compilamos
> ChdsMCMC <- compileNimble(hdsMCMC, project = Rmodel)
> # Ejecutamos el modelo
> nb=5000      # Iteraciones a desechar
> ni=2500 +nb  # Iteraciones
> nc=3         # Cadenas
```



```
> start.time2<-Sys.time()
> outNim <- runMCMC(ChdsMCMC, 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 1.997509 secs

Observamos resultados

```
> summary(outNim)
```

Iterations = 1:2500

Thinning interval = 1

Number of chains = 3

Sample size per chain = 2500

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

	Mean	SD	Naive SE	Time-series SE
D	0.1483	0.01793	0.0002071	0.0009758
Ntotal	148.3289	17.93222	0.2070634	0.9757663
alpha0	0.8714	0.12344	0.0014254	0.0063474
beta0	2.6848	0.14635	0.0016899	0.0076251

2. Quantiles for each variable:

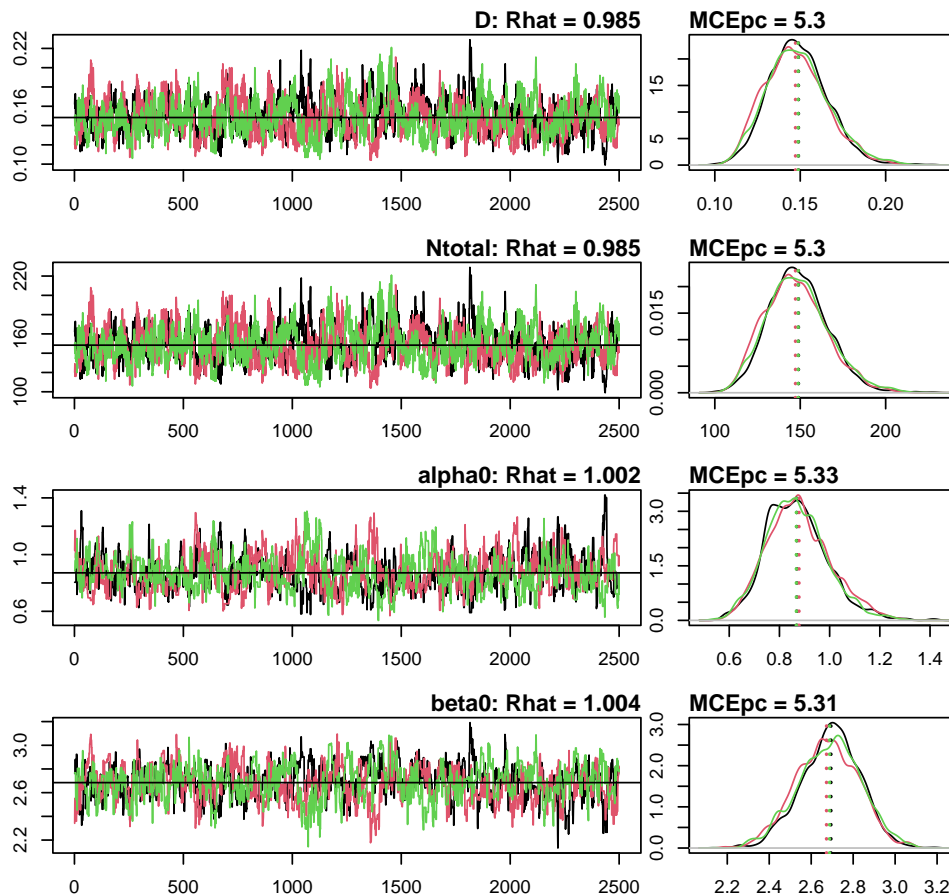
	2.5%	25%	50%	75%	97.5%
D	0.1170	0.1360	0.1470	0.1600	0.186

```
Ntotal 117.0000 136.0000 147.0000 160.0000 186.000
alpha0  0.6538  0.7836  0.8638  0.9484  1.145
beta0    2.3925  2.5868  2.6908  2.7877  2.963
```

...y comprobamos convergencia

```
> library(mcmcOutput)
> diagPlot(mcmcOutput(outNim))
```

Diagnostics for mcmcOutput(outNim)



4. REFERENCIAS

- Burnham, K. P., Anderson, D. R., & Laake, J. L. (1980). Estimation of density from line transect sampling of biological populations. *Wildlife Monographs*, 72(72), 3–202. doi:10.1126/science.98.2539.185
- 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
- Fiske, I. J., & Chandler, R. B. (2011). unmarked: An R package for fitting hierarchical models of wildlife occurrence and abundance. *Journal of Statistical Software*, 43(10), 1–23. doi:10.1002/wics.10
- Kéry, M., & Royle, J. A. (2016). *Applied Hierarchical Modeling in Ecology Analysis of distribution, abundance and species richness in R and BUGS* (1st ed., Vol. 1). Academic Press / Elsevier.
- 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/>.