01.2_PhotoID_CMR_Bayes

2024-10-28

Capture mark recapture of Arctic killer whales using a genetic (whole genome) identification history. Here, we will analyze the genetic CMR data using a Bayesian framework, as described in Marc and Kéry (2011).

The script was obtained from https://github.com/oliviergimenez/bayes-multistate-jollyseber, modified for these data (described in 02.2_PhotoID_CMR_Bayes).

POPAN Jolly Seber

Assumptions for Jolly-Seber Mark Recapture models: 1. Animals retain their tags throughout the experiment 2. Tags are read properly 3. Sampling is instantaneous 4. Survival probabilities are the same for all animals (marked and unmarked) between each pair of sampling occasions (homogenous survival) 5. Catchability is the same for all animals (marked and unmarked) at each sampling occasion (homogenous catchability) 6. The study area is constant

In the case of killer whales, do we meet the assumptions? 1. Yes - most nicks and scars used for ID are retained through the life of the animal. 2. Yes - we can assume that identified individuals are re-identified reliably. However, there are ways to account for identification error - to be explored later. 3. Yes - sampling period is short (1 to a few days) 4. Probably? - while survival probabilities might differ between sex and age classes, being "marked" does not affect an individual's survival 5. Unlikely? - Equal catchability could be affected by: - Behaviour: some individuals/groups may be more likely to approach the boat, and thus we may get more/better photographs - Individuals with more distinct markings may be more likely to be identified/re-identified when image quality is lower - Cooch and White (2014) describe this as the most ciritical assumption for JS models 6. Sort of? - The study area is confined to locations around Northern Baffin Island (mainly Admiralty Inlet and Eclipse Sound) and Cumberland Sound, but we have not consistently sampled in each location each year

Analysis

Prep the environment:

See 02.2_PhotoID_CMR_Bayes for description of model.

POPAN parameterization of data:

```
popan <- function() {

   # Priors and constraints
  for (i in 1:M){
    for (t in 1:(n.occasions-1)){
      phi[i,t] <- mean.phi  # Constant survival
    } #t
    for (t in 1:n.occasions){
      p[i,t] <- p.time[t]  # Time-dependent capture
    } #t
  } #i

  mean.phi ~ dunif(0, 1)  # Prior for mean survival - uniform distribution with min = 0 and max = 1
  psi ~ dunif(0, 1) # Prior for inclusion probability
```

```
for(t in 1:n.occasions) {
  p.time[t] ~ dunif(0,1)  # prior for time-dependent capture
}

# Dirichlet prior for entry probabilities
for (t in 1:n.occasions){
  beta[t] ~ dgamma(1, 1)  # gamma distribution with shape = 1 and scale = 1
  b[t] <- beta[t] / sum(beta[1:n.occasions])
}

# Convert entry probs to conditional entry probs
nu[1] <- b[1]
for (t in 2:n.occasions){
  nu[t] <- b[t] / (1 - sum(b[1:(t-1)]))
} #t

# Likelihood
for (i in 1:M){
  # First occasion
  # State process
  w[i] ~ dbern(psi)  # Draw latent inclusion
  z[i,1] ~ dbern(nu[1])
  # Observation process
  mu1[i] <- z[i,1] * p[i,1] * w[i]
  y[i,1] ~ dbern(mu1[i])
  # Subsequent occasions
  for (t in 2:n.occasions){
    # State process
    q[i,t-1] <- 1 - z[i,t-1]
    mu2[i,t] <- phi[i,t-1] * z[i,t-1] + nu[t] * prod(q[i,1:(t-1)])
    z[i,t] ~ dbern(mu2[i,t])
    # Observation process
    mu3[i,t] <- z[i,t] * p[i,t] * w[i]
    y[i,t] ~ dbern(mu3[i,t])
  } #t
} #i

# Calculate derived population parameters
for (i in 1:M){
  for (t in 1:n.occasions){
    u[i,t] <- z[i,t] * w[i]  # Deflated latent state (u)
  }
}
for (i in 1:M){
  recruit[i,1] <- u[i,1]
  for (t in 2:n.occasions){
    recruit[i,t] <- (1 - u[i,t-1]) * u[i,t]
  } #t
} #i
for (t in 1:n.occasions){
  N[t] <- sum(u[1:M,t])  # Actual population size
  B[t] <- sum(recruit[1:M,t])  # Number of entries
} #t
```

```
  for (i in 1:M){
    Nind[i] <- sum(u[i,1:n.occasions])
    Nalive[i] <- 1 - equals(Nind[i], 0)
  } #i
  for (t in 1:(n.occasions-1)) {
    lambda[t] <- N[t+1]/N[t] # Lambda realized annual population growth rate
    f[t] <- B[t+1]/N[t] # recruitment (per capita entry) rate
  } #t
  Nsuper <- sum(Nalive[]) # Superpopulation size
  mean.lambda <- (prod(lambda[]))^(1/(n.occasions-1)) # geometric mean realized growth rate
}
```

Now let's apply it to our data.

Load and format the genetic capture history data:

```
# Load data and insert zeros instead of NAs in years where there are no sightings:
CMR_data_gen <- read.csv("genetic_CMR_data_final.csv", header = TRUE)
CMR_data_gen[is.na(CMR_data_gen)] <- 0

# Remove the id column and add a new column for CH data:
CMR_data_gen <- CMR_data_gen %>%
  select(-genome_sample_ID) %>%
  mutate(cmr = NA) %>%
  relocate(cmr, "X2013")

# Fill the cmr column with 1s and 0s for all years observed:
for (i in 1:nrow(CMR_data_gen)){
  CMR_data_gen[i,1] <- paste(CMR_data_gen[i,2], CMR_data_gen[i,3], CMR_data_gen[i,4], CMR_data_gen[i,5]
                             CMR_data_gen[i,6], CMR_data_gen[i,7], CMR_data_gen[i,8], CMR_data_gen[i,9]
                             CMR_data_gen[i,10], CMR_data_gen[i,11],sep = "")}

# Select first column only, rename, and save to txt file
CMR_gen <- CMR_data_gen %>%
  select(cmr) %>%
  dplyr::rename(ch = cmr) %>%
  write.table(file = "_genetic_cmr_data.txt", row.names = FALSE, quote = FALSE)

# Re-import and split data
ch_CMR_gen <- import.chdata("_genetic_cmr_data.txt")
popan_ch_gen <- splitCH(ch_CMR_gen$ch)
```

Augment the observed capture histories by nz pseudo-individuals, all with capture histories of 0:

```
nz <- 200     # Augmenting the data by 200 pseudo-individuals
CH.aug.gen <- rbind(popan_ch_gen, matrix(0, ncol = dim(popan_ch_gen)[2], nrow = nz))
```

Bundle data.

```
bugs.data.gen <- list(y = CH.aug.gen,
                      n.occasions = dim(CH.aug.gen)[2],
                      M = dim(CH.aug.gen)[1])
```

Initial values.

```
zinit <- CH.aug.gen
zinit[zinit==0] <- 1
```

```r
n.occasions.js <- ncol(CH.aug.gen)

inits <- function(){list(mean.phi = runif(1, 0, 1),
                         p.time = runif(n.occasions.js, 0, 1),
                         psi = runif(1, 0, 1),
                         z = zinit)}
```

Parameters monitored.

```r
parameters <- c("psi", "p.time", "mean.phi", "b", "Nsuper", "N", "B", "nu", "lambda", "f", "mean.lambda
```

MCMC settings.

```r
n.iter   <- 50000     # Number of iterations
n.burnin <- 10000     # Number discarded (burn-in)
n.chains <- 3         # Number of chains
```

Call Jags - run model on bio server.

```r
# kw_popan_gen <- R2jags::jags(data  = bugs.data.gen,
#                       inits = inits,
#                       parameters.to.save = parameters,
#                       model.file = popan,
#                       n.chains = n.chains,
#                       n.iter = n.iter,
#                       n.burnin = n.burnin)
# kw_popan_gen
```

Save run:

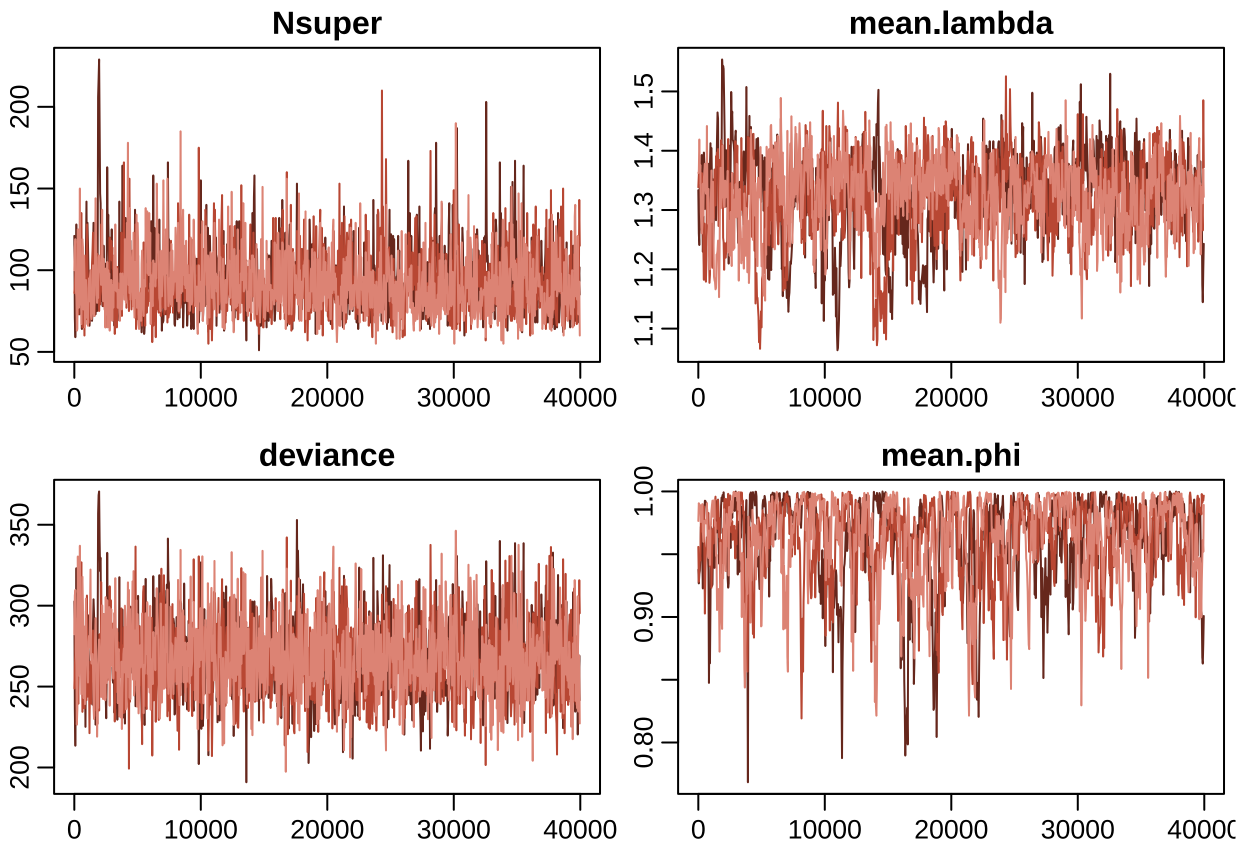
```r
#saveRDS(kw_popan_gen, "output/kw_popan_gen_results_sept2024_run1.rds")

kw_popan_gen <- readRDS("output/kw_popan_gen_results_sept2024_run1.rds")
kw_popan_gen
```

```
## Inference for Bugs model at "/var/folders/5d/d_4b4fhd47l0w3nzrwbh807h0000gn/T//RtmpoZBmMr/modelad7364
##  3 chains, each with 50000 iterations (first 10000 discarded), n.thin = 40
##  n.sims = 3000 iterations saved
##             mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
## B[1]          9.356   4.751   6.000   6.000   8.000  10.000  23.000 1.004  2600
## B[2]          5.354   7.396   0.000   1.000   3.000   7.000  27.000 1.058   230
## B[3]          6.383   7.711   0.000   1.000   4.000   9.000  27.000 1.010   290
## B[4]          8.899   8.782   0.000   2.000   6.000  13.000  31.000 1.019   200
## B[5]         12.845  11.763   0.000   4.000  10.000  19.000  42.025 1.014   160
## B[6]         18.642  15.571   0.000   7.000  15.000  27.000  56.025 1.022   220
## B[7]         14.811  12.563   0.000   5.000  12.000  21.000  47.000 1.030   150
## B[8]         11.270  10.201   0.000   3.000   9.000  16.000  38.000 1.039    89
## B[9]          5.420   6.314   0.000   1.000   3.000   8.000  22.000 1.011   630
## N[1]          9.356   4.751   6.000   6.000   8.000  10.000  23.000 1.004  2600
## N[2]         14.474   9.201   6.000   8.000  12.000  17.000  40.000 1.011  2600
## N[3]         20.423  12.412   7.000  11.000  17.000  26.000  53.000 1.006   760
## N[4]         28.664  14.497   8.000  18.000  26.000  37.000  63.000 1.012   560
## N[5]         40.580  16.629  13.000  29.000  39.000  51.000  77.000 1.018   180
## N[6]         57.955  16.893  30.000  46.000  56.000  68.000  95.025 1.019   120
## N[7]         70.707  16.162  45.000  60.000  68.000  80.000 108.000 1.008   280
## N[8]         79.323  16.631  54.000  68.000  77.000  88.000 118.025 1.002  1500
## N[9]         81.806  18.978  53.000  69.000  79.000  91.000 126.000 1.004  1100
## Nsuper       92.980  19.849  64.000  79.000  90.000 103.000 139.000 1.003   800
## b[1]          0.105   0.058   0.033   0.066   0.092   0.129   0.263 1.002  3000
## b[2]          0.063   0.075   0.001   0.014   0.037   0.080   0.282 1.004  3000
## b[3]          0.073   0.075   0.002   0.019   0.047   0.101   0.277 1.011   200
## b[4]          0.096   0.087   0.002   0.027   0.070   0.144   0.303 1.009   300
## b[5]          0.136   0.115   0.004   0.045   0.105   0.199   0.416 1.018   130
## b[6]          0.194   0.150   0.007   0.072   0.160   0.285   0.560 1.004   580
## b[7]          0.153   0.118   0.005   0.058   0.125   0.226   0.432 1.004   530
## b[8]          0.121   0.097   0.004   0.043   0.100   0.172   0.362 1.011   190
## b[9]          0.059   0.055   0.001   0.018   0.044   0.084   0.200 1.004   640
## f[1]          0.626   0.939   0.000   0.098   0.333   0.778   3.375 1.049   360
## f[2]          0.534   0.712   0.000   0.100   0.294   0.700   2.500 1.027   200
## f[3]          0.587   0.726   0.000   0.111   0.333   0.786   2.501 1.012   300
## f[4]          0.640   0.835   0.000   0.128   0.353   0.808   3.100 1.001  2100
## f[5]          0.695   0.996   0.000   0.143   0.375   0.833   3.381 1.059   160
## f[6]          0.312   0.337   0.000   0.080   0.201   0.429   1.269 1.043   110
## f[7]          0.178   0.183   0.000   0.046   0.125   0.250   0.702 1.046    85
## f[8]          0.069   0.077   0.000   0.013   0.045   0.099   0.282 1.009   870
## lambda[1]     1.604   0.943   0.889   1.043   1.286   1.762   4.334 1.014  1400
## lambda[2]     1.508   0.714   0.917   1.067   1.275   1.688   3.444 1.015   170
## lambda[3]     1.558   0.729   0.926   1.083   1.308   1.769   3.500 1.010   290
## lambda[4]     1.610   0.840   0.946   1.100   1.320   1.778   4.084 1.003   760
## lambda[5]     1.664   0.997   0.968   1.113   1.345   1.800   4.364 1.026   210
## lambda[6]     1.277   0.338   0.945   1.049   1.167   1.390   2.234 1.029   130
## lambda[7]     1.140   0.183   0.920   1.015   1.090   1.209   1.632 1.035    88
## lambda[8]     1.030   0.088   0.892   0.980   1.013   1.069   1.247 1.006  2100
## mean.lambda   1.323   0.070   1.168   1.283   1.332   1.371   1.438 1.005  2700
## mean.phi      0.962   0.034   0.873   0.946   0.970   0.987   0.999 1.007  3000
## nu[1]         0.105   0.058   0.033   0.066   0.092   0.129   0.263 1.002  3000
## nu[2]         0.071   0.085   0.001   0.015   0.041   0.091   0.321 1.004  3000
## nu[3]         0.090   0.094   0.002   0.023   0.057   0.129   0.334 1.009   230
## nu[4]         0.128   0.114   0.003   0.037   0.095   0.191   0.403 1.009   320
## nu[5]         0.205   0.163   0.006   0.074   0.167   0.301   0.590 1.018   130
## nu[6]         0.353   0.224   0.019   0.166   0.327   0.527   0.804 1.009   250
## nu[7]         0.441   0.246   0.026   0.237   0.436   0.640   0.891 1.002  1300
## nu[8]         0.635   0.275   0.056   0.435   0.685   0.874   0.992 1.002  1200
## nu[9]         1.000   0.000   1.000   1.000   1.000   1.000   1.000 1.000     1
## p.time[1]     0.687   0.218   0.234   0.533   0.723   0.872   0.986 1.002  3000
## p.time[2]     0.075   0.079   0.001   0.018   0.048   0.103   0.299 1.003  1000
## p.time[3]     0.057   0.065   0.001   0.014   0.036   0.076   0.251 1.002  1200
## p.time[4]     0.041   0.048   0.001   0.010   0.025   0.052   0.176 1.003  1500
## p.time[5]     0.028   0.037   0.001   0.007   0.017   0.036   0.131 1.003   870
## p.time[6]     0.162   0.070   0.062   0.111   0.150   0.197   0.336 1.004   660
## p.time[7]     0.159   0.055   0.072   0.118   0.152   0.191   0.283 1.004   500
## p.time[8]     0.321   0.081   0.176   0.263   0.316   0.370   0.497 1.002  1600
## p.time[9]     0.100   0.040   0.037   0.072   0.095   0.121   0.195 1.004   590
## psi           0.381   0.086   0.246   0.321   0.369   0.427   0.570 1.003   680
## deviance    267.703  23.836 224.542 250.777 266.542 282.411 317.858 1.001  3000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 284.1 and DIC = 551.8
## DIC is an estimate of expected predictive error (lower deviance is better).
```

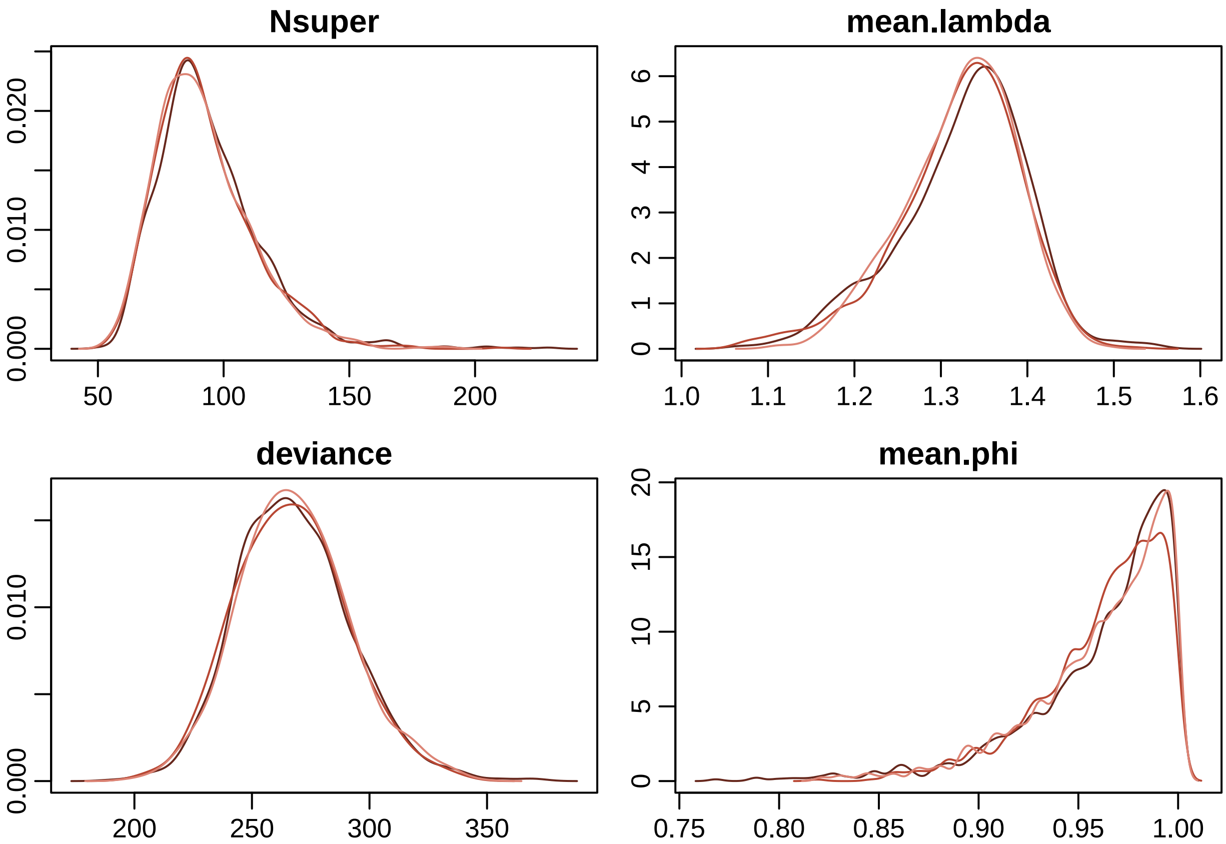
```
library(mcmcplots)
```

```
## Registered S3 method overwritten by 'mcmcplots':
##   method        from
##   as.mcmc.rjags R2jags
```

```
library(coda)

# Convert the results to an mcmc list object
kw_popan_gen.mcmc <- as.mcmc(kw_popan_gen)

# Save mcmc object:
#saveRDS(kw_popan_gen.mcmc, "output/kw_popan_gen_mcmc_sept2024_run1.rds")

cols.gen <- c("#66271c", "#b84733", "#dc8374")

# Traceplot
#tiff("plots/kw_popan_gen_traceplot.tiff", units="in", width=8, height=5, res=400)
mcmcplots::traplot(kw_popan_gen.mcmc, parms = c("Nsuper", "mean.lambda", "deviance", "mean.phi"), style
```



```
#dev.off()

# Density plot
#tiff("plots/kw_popan_gen_densityplot.tiff", units="in", width=8, height=5, res=400)
mcmcplots::denplot(kw_popan_gen.mcmc, parms = c("Nsuper", "mean.lambda", "deviance", "mean.phi"), style
```


Nsuper

mean.lambda

deviance

mean.phi

```
#dev.off()
```