Spatial close-kin mark-recapture models applied to terrestrial species with continuous natal dispersal.

Appendix S2. Step-by-step guide on implementing spatial CKMR models to real data.

Any use of trade, firm or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

1. Setting up the stage

1a. Load packages, source functions, and demographic parameters

```r
# Packages
library(tidyverse) # Data management
library(nimble) # Run models
library(MCMCvis) # Visualize models
library(parallel) # For parallel processing of MCMCs
library(sf) # Spatial stuff
library(terra) # Spatial stuff
library(tidyterra) # To plot SpatRaster with ggplot
library(nimbleDistance) # To obtain dispersal pdf
library(ggplot2) # For plots

# Source functions
source("./functions/pairwise_comparison.R")
source("./functions/spatial_parameters.R")

# Demographic parameters
repro.age <- 2 # Age of reproductive maturity
max.age <- 10 # Maximum age
t <- 28 # Year of estimation
my.crs <- 32616 # EPSG code or full name
state.space <- st_read("./GIS/UP_Michigan_mainland.shp", quiet = TRUE) %>%
    st_as_sfc(crs = 4326) # State space boundary
```

Important
If your data is already projected (with `my.crs`), you need to change the crs of `state.space` and `samples` to WGS84. All the functions are using `lon` and `lat` columns to infer spatial parameters. We will project the features to `my.crs` later in the functions. You also need to make sure that `my.crs` has unit = meter.
Some of the functions used below may need to be adjusted to applications that include more covariates in the kinship probabilities (e.g., fecundity at age).

1b. Load samples

Let us load "realistically fake" samples with the minimum amount of information required to run spatial CKMR (also excluding the genetic data). We are loosely mimicking an American black bear (*Ursus americanus*) population in the upper peninsula of Michigan (USA). In this population, females are mostly philopatric (mean dispersal = 15km) whilst males disperse longer (mean dispersal = 50km). There is a higher density of bears in the western half of the peninsula, and sampling is spatially biased (samples were only collected in the four largest counties).

Note: the parameters used for this simulation have little support in real life, and should not be extrapolated to infer anything on the real black bear population of Michigan. This is for illustrative purposes only.

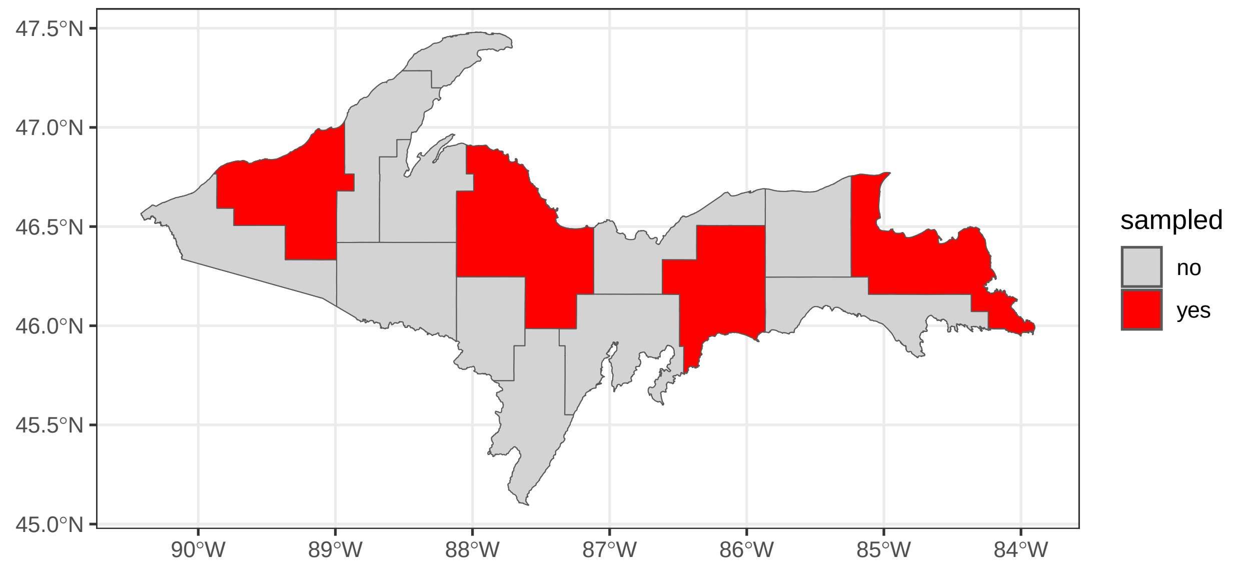
```
load("./data/michigan_fake_samples.RData")
```

The `samples` dataframe has the following columns:
- *ID:* unique ID
- *birth.year:* year of birth of each sampled individual, estimated with ageing methods
- *sampling.age:* age of each individual when sampled
- *sex:* sex
- *lon:* longitude of sampling location
- *lat:* latitude of sampling location
- *sampling.year:* year of sampling

```
## 'data.frame':    1044 obs. of  7 variables:
##  $ ID           : chr  "arwtvnaqkawmolyhimyw" "lhsoyylyvhkkdmejliax" "ngrnpqjhsnzbqovvcxge" ...
##  $ birth.year   : num  18 18 18 18 18 19 19 19 ...
##  $ sampling.age : num  10 10 10 10 10 10 10 10 ...
##  $ sex          : chr  "F" "M" "M" ...
##  $ lon          : num  -87.9 -84.5 -89 -86.4 ...
##  $ lat          : num  46.7 46.2 46.6 46.2 ...
##  $ sampling.year: num  28 28 28 28 28 29 29 29 ...
```

Let's also load a dataframe with the mother-offspring pairs (MOP) that would be identified through the genotyping process. The `MOP` dataframe has the following columns:
- *Ind_1:* mother (individual i)
- *Ind_1_sex:* sex of the parent
- *Ind_2:* offspring (individual j)
- *Ind_2_sex:* sex of the offspring
- *KinPair:* kinship relationship. Either Mom-Son, Mom-Daughter, or Unrelated (UR).

```
## 'data.frame':    181 obs. of  5 variables:
##  $ Ind_1    : chr  "tajglzvzmsbqxqrhcmwu" "pqvskuxdwqwrmpbtvbtj" "arwtvnaqkawmolyhimyw" ...
##  $ Ind_1_sex: chr  "F" "F" "F" ...
##  $ Ind_2    : chr  "jkowjohbzmfabrftumya" "sdcyizgcnmqxlrqvogqc" "hnnjsbtltgwgciqpanfp" ...
##  $ Ind_2_sex: chr  "F" "F" "F" ...
##  $ KinPair  : chr  "MomDau" "MomDau" "MomDau" ...
```



2. Create pairwise comparisons

First, we need to create a dataframe with all *i* x *j* pairwise comparisons possible. We trim these pairwise comparisons to retain only those that can be a mother-offspring pair (i.e., *i* must be a female alive and mature when *j* was born). This step uses the demographic parameters specified above. We also calculate the Euclidean distance (m) between the two individuals in each pair, and we import the kinship information from the `MOP` dataframe. We run all this with the `build.pairwise.spatial.mop` function.

```
pop_mom_pairs <- build.pairwise.spatial.mop (samples = samples)
```

```
##                    Ind_1 Ind_1_sex Ind_1_birth Ind_1_sampling Ind_1_lon Ind_1_lat
## 1 arwtvnaqkawmolyhimyw         F          18             28 -87.94066  46.65127
## 2 rtcqhskumpwzrluejoak         F          18             28 -86.37773  46.18580
## 3 nijgfbmwidqcxrottuoa         F          18             28 -87.62512  46.72892
## 4 arwtvnaqkawmolyhimyw         F          18             28 -87.94066  46.65127
## 5 nijgfbmwidqcxrottuoa         F          18             28 -87.62512  46.72892
##                    Ind_2 Ind_2_sex Ind_2_birth Ind_2_sampling Ind_2_lon Ind_2_lat
## 1 ojnuksbbiyvtsenhihwe         F          20             28 -86.44173  46.26407
## 2 ojnuksbbiyvtsenhihwe         F          20             28 -86.44173  46.26407
## 3 ojnuksbbiyvtsenhihwe         F          20             28 -86.44173  46.26407
## 4 kpdfosjcfjtmpuaxsuwq         M          20             28 -84.42800  46.31444
## 5 kpdfosjcfjtmpuaxsuwq         M          20             28 -84.42800  46.31444
##     distance Kin
## 1 122623.866  UR
## 2   9999.463  UR
## 3 104291.861  UR
## 4 271521.455  UR
## 5 248900.080  UR
```

Separate MOP comparisons by offspring sex, and check how many mother-daughter and mother-son pairs we have.

```
mom_daughter <- pop_mom_pairs[pop_mom_pairs$Ind_2_sex == "F",]
mom_son <- pop_mom_pairs[pop_mom_pairs$Ind_2_sex == "M",]

MomDau <- as.numeric(count(pop_mom_pairs[pop_mom_pairs$Kin == "MomDau",]))
MomSon <- as.numeric(count(pop_mom_pairs[pop_mom_pairs$Kin == "MomSon",]))
```

```
## [1] "MomDau = 123"
```

```
## [1] "MomSon = 58"
```

3. Spatial parameters

3a. Natal range probability (p_{ij})

Calculate the natal range probability for each pairwise comparisons, based on distances. We use the `pij` function.

```
dispersal <- pij (pop_mom_pairs = pop_mom_pairs)
```

Retrieve the `pop_mom_pairs` dataframe, with now p_{ij} added. We also save the parameters of the hazard-rate function for male and female dispersal (which we need in the next step).

```
pop_mom_pairs <- dispersal[[1]]

dHR_F_sigma <- dispersal[[2]]
dHR_F_b <- dispersal[[3]]
dHR_M_sigma <- dispersal[[4]]
dHR_M_b <- dispersal[[5]]
disp.max.F <- dispersal[[6]]
disp.max.M <- dispersal[[7]]
```

```
##                    Ind_1 Ind_1_sex Ind_1_birth Ind_1_sampling Ind_1_lon
## 1  arwtvnaqkawmolyhimyw         F          18             28 -87.94066
## 2  rtcqhskumpwzrluejoak         F          18             28 -86.37773
## 3  nijgfbmwidqcxrottuoa         F          18             28 -87.62512
## 10 nijgfbmwidqcxrottuoa         F          18             28 -87.62512
## 11 rtcqhskumpwzrluejoak         F          18             28 -86.37773
##    Ind_1_lat                Ind_2 Ind_2_sex Ind_2_birth Ind_2_sampling
## 1   46.65127 ojnuksbbiyvtsenhihwe         F          20             28
## 2   46.18580 ojnuksbbiyvtsenhihwe         F          20             28
## 3   46.72892 ojnuksbbiyvtsenhihwe         F          20             28
## 10  46.72892 ddotbcslscvuriydtaqe         F          20             29
## 11  46.18580 ddotbcslscvuriydtaqe         F          20             29
##    Ind_2_lon Ind_2_lat   distance Kin   pij
## 1  -86.44173  46.26407 122623.866  UR 0.001
## 2  -86.44173  46.26407   9999.463  UR 1.000
## 3  -86.44173  46.26407 104291.861  UR 0.001
## 10 -86.45347  45.80307 136782.687  UR 0.001
## 11 -86.45347  45.80307  42958.150  UR 0.120
```

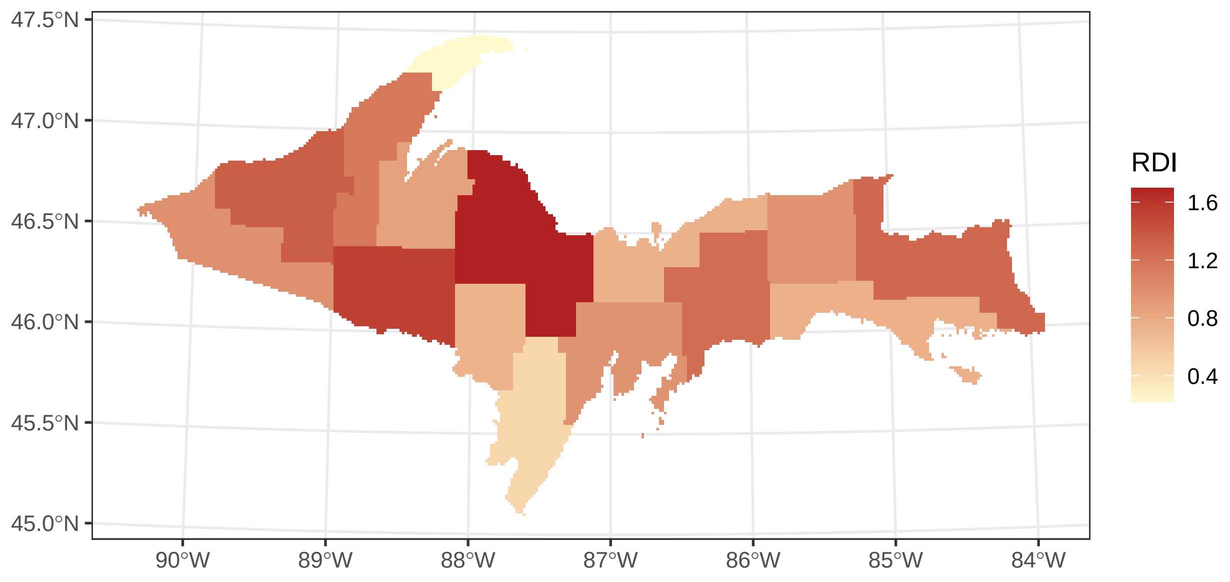
3b. Sampled area α_j and $\alpha_{j,D}$

We load the auxiliary data that will be used as the relative density index (D in the manuscript; *RDI* below). Here, we load two rasters: the first raster (`RDI.broad`) has one estimate of relative density per county; the second raster (`RDI.fine`) consists of 10x10km cells. Both rasters are without error, and must already be projected to `my.crs`. Remember, the relative density index D must be centred around mean density, and scaled so that mean density = 1. Any value higher than 1 shows high relative density, and any value lower than 1 shows low relative density.

Note: what if we want to run the model but we do not have a relative density index raster (e.g., because we know the population is relatively homogeneous)? One solution could be to create a raster fitted to the dimensions of `state.space` (and with crs = `my.crs`), and fill it with cell values = 1. This way, there is no need to modify the `kj` and `build.grouped.spatial.mop` functions.

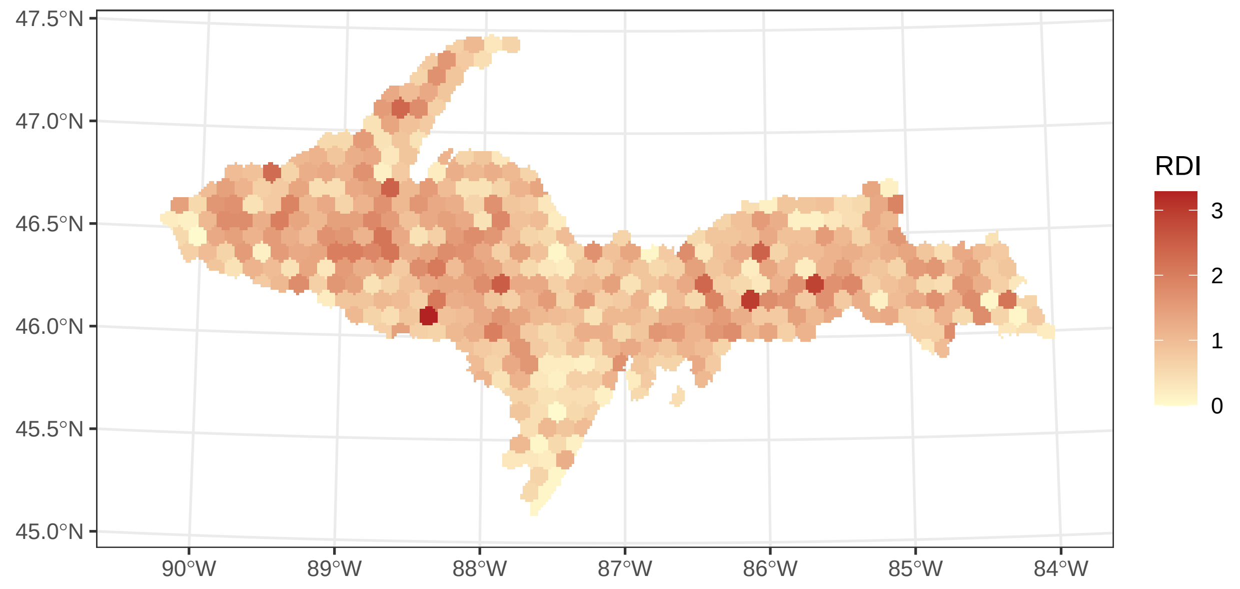
Broad raster:

```r
RDI.broad <- terra::rast("./data/michigan_broad_RDI.tif")
```



Fine raster:

```r
RDI.fine <- terra::rast("./data/michigan_fine_RDI.tif")
```



And now, we run the function `kj` to obtain the scaling factors k_j and $k_{j,D}$ for each pairwise comparisons, which are obtained by the ratio between the area of the state space S and α_{j}, and S and $\alpha_{\text{j,D}}$, respectively. We could also add this ratio directly into to the kinship probabilities (as shown in the manuscript), but this would require us to run each pairwise comparisons sequentially (as opposed to grouping comparisons; see below). Here, you can change the `density.pop.raster` to `RDI.fine` or `RDI.broad` to compare estimates of population size.

Note: this step takes a bit of time… we can go grab a tea or coffee.

```
pop_mom_pairs <- kj(pop_mom_pairs = pop_mom_pairs,
                    mom_daughter = mom_daughter,
                    mom_son = mom_son,
                    density.pop.raster = RDI.fine,
                    state.space = state.space,
                    my.crs = my.crs)
```

```
##                   Ind_2                Ind_1 Ind_1_sex Ind_1_birth
## 1 aafbryyyxfyyhgmeksti qecgyhpdelvnnswxogph         F          25
## 2 aafbryyyxfyyhgmeksti rfhhhhtabuudfediikdg         F          25
## 3 aafbryyyxfyyhgmeksti yxxadvyraqunqbueryri         F          25
## 4 aafbryyyxfyyhgmeksti tvxdnufjgxesbuhuubwk         F          25
## 5 aafbryyyxfyyhgmeksti dxdhwthnnhkljdjmsbeg         F          25
##   Ind_1_sampling Ind_1_lon Ind_1_lat Ind_2_sex Ind_2_birth Ind_2_sampling
## 1             28 -87.74358  46.57478         M          27             30
## 2             28 -89.37734  46.63043         M          27             30
## 3             28 -89.24038  46.47731         M          27             30
## 4             28 -87.87407  46.57386         M          27             30
## 5             30 -86.38607  45.87459         M          27             30
##   Ind_2_lon Ind_2_lat distance Kin   pij   kj kj_D
## 1 -85.20684  46.32932 196236.5  UR 0.001 4.42 4.24
## 2 -85.20684  46.32932 321047.5  UR 0.001 4.42 4.24
## 3 -85.20684  46.32932 309686.7  UR 0.001 4.42 4.24
## 4 -85.20684  46.32932 206127.4  UR 0.001 4.42 4.24
## 5 -85.20684  46.32932 104031.8  UR 0.060 4.42 4.24
```

4. Create grouped comparisons dataframe

Now that we have all the information necessary for each pairwise comparison, we can group comparisons that have the same combination of covariates with the function `build.grouped.spatial.mop` to optimize computational efficiency. Because there are 5 covariates here (birth year of i; birth year of j, p_{ij}, k_j and $k_{j,D}$), there are still a lot of unique comparisons left, but we still managed to drastically diminish the number of pairwise comparison.

```
pop_mom_comps <- build.grouped.spatial.mop(pop_mom_pairs = pop_mom_pairs)
```

```
##   Ind_1_birth Ind_2_birth pij    kj  kj_D yes no all
## 1          18          25   1 11.53  9.66   1  0   1
## 2          18          26   1  2.99  2.73   1  5   6
## 3          18          27   1  4.11  4.13   1  0   1
## 4          18          27   1 19.51 22.08   1  3   4
## 5          18          28   1 20.61 22.28   1  1   2
```

5. CKMR model

And at last, we run the CKMR model. Here we use Bayesian inference, but the same models can be implemented with maximum likelihood estimation. We run four chains in parallel.
Nf1 refers to the naïve model.
Nf2 refers to the spatial model without derived total abundance.
Nf3 refers to the spatial model with derived total abundance but without the relative density index.
Nf4 refers to the spatial model with derived total abundance and with the relative density index.

We create the core cluster for parallelisation (one core per MCMC).

```
this_cluster <- makeCluster(4)
```

We create the data and constants bundles.

```
 # Data bundle
my.data <- list(MatPOP1 = pop_mom_comps$yes, # Positive maternal parent offspring
                pop_mom_all_comps1 = pop_mom_comps$all, # Total maternal pop comparisons
                MatPOP2 = pop_mom_comps$yes,
                pop_mom_all_comps2 = pop_mom_comps$all,
                MatPOP3 = pop_mom_comps$yes,
                pop_mom_all_comps3 = pop_mom_comps$all,
                MatPOP4 = pop_mom_comps$yes,
                pop_mom_all_comps4 = pop_mom_comps$all)

  # Constants bundle
my.constants <- list(pop_mom_length = nrow(pop_mom_comps), # Total number of comparisons
                     j_birthyear = pop_mom_comps$Ind_2_birth,
                     t = t,
                     pij_mom = pop_mom_comps$pij,
                     kj_mom = pop_mom_comps$kj,
                     kj_D_mom = pop_mom_comps$kj_D)
```

We specify the parameters for the model. These includes priors, kinship likelihoods, derived population dynamics, chains parameters, and the syntax of the nimbleModel.

```
myCode <- nimbleCode({

    # priors
    Nf1 ~ dunif(1, 1e4) # Uniform prior for female abundance
    Nf2 ~ dunif(1, 1e4)
    Nf3 ~ dunif(1, 1e4)
    Nf4 ~ dunif(1, 1e4)

    r1 ~ dunif(-0.5, 0.5) # Uniform prior for population growth.
    r2 ~ dunif(-0.5, 0.5)
    r3 ~ dunif(-0.5, 0.5)
    r4 ~ dunif(-0.5, 0.5)

    # likelihoods

    # Naive model
    for (i in 1:pop_mom_length){
      MatPOP1[i] ~ dbinom(
        1 / (Nf1 * exp(r1 * (j_birthyear[i] - t))),
        pop_mom_all_comps1[i])
    }

    # Without derivation for total abundance
    for (j in 1:pop_mom_length){
      MatPOP2[j] ~ dbinom(
        pij_mom[j] / (Nf2 * exp(r2 * (j_birthyear[j] - t))),
        pop_mom_all_comps2[j])
    }

    # N total derived without relative density
    for (k in 1:pop_mom_length){
      MatPOP3[k] ~ dbinom(
        pij_mom[k] / ( Nfxj3[k] * exp(r3 * (j_birthyear[k] - t))),
        pop_mom_all_comps3[k])

      Nfxj3[k] <- Nf3 / kj_mom[k]
    }

    # N total derived with relative density
    for (l in 1:pop_mom_length){
      MatPOP4[l] ~ dbinom(
        pij_mom[l] / (Nfxj4[l] * exp(r4 * (j_birthyear[l] - t))),
        pop_mom_all_comps4[l])

      Nfxj4[l] <- Nf4 / kj_D_mom[l]
```

```r
  }
})


# Create a function with all the needed code
run_MCMC_allcode <- function(seed, data, constants, code) {

  library(nimble) # Not sure why, but this needs to be specified in the function

  myModel <- nimbleModel(code = code,
                         data = data,
                         constants = constants,
                         inits = list(Nf1 = runif(1, 100, 1e4),
                                      Nf2 = runif(1, 100, 1e4),
                                      Nf3 = runif(1, 100, 1e4),
                                      Nf4 = runif(1, 100, 1e4),
                                      r1 = rnorm(1, mean = 0, sd = 0.05),
                                      r2 = rnorm(1, mean = 0, sd = 0.05),
                                      r3 = rnorm(1, mean = 0, sd = 0.05),
                                      r4 = rnorm(1, mean = 0, sd = 0.05)))

  # Chain parameters
  n.iter <- 50000
  n.burnin <- 10000
  n.thin <- 10

  CmyModel <- compileNimble(myModel)
  myMCMC <- buildMCMC(CmyModel)
  CmyMCMC <- compileNimble(myMCMC)

  results <- runMCMC(CmyMCMC,
                     niter = n.iter, nburnin = n.burnin, thin = n.thin,
                     setSeed = seed)

  return(results)
}
```

And we run the model!

```r
chain_output <- parLapply(cl = this_cluster, X = 1:4, # x = 1:n.chains
                          fun = run_MCMC_allcode,
                          data = my.data,
                          constants = my.constants,
                          code = myCode)
```

It is good practice to close the cluster once you are done with it.

```r
stopCluster(this_cluster)
```

And we look at the results! True adult female population size for estimation year was Nf = 924. Estimates of r refer to the average annual population rate of increase.

```r
MCMCsummary <- MCMCsummary(chain_output, round = 3)
```

```
##         mean     sd    2.5%     50%    97.5% Rhat n.eff
## Nf1 617.032 48.355 530.045 614.490  719.591    1 15616
## Nf2 124.573  9.764 106.922 124.005  144.951    1 15287
## Nf3 951.620 74.769 816.638 948.497 1107.948    1 15299
## Nf4 900.175 70.231 774.040 896.248 1050.344    1 14668
## r1   -0.083  0.072  -0.226  -0.082    0.056    1 15311
## r2   -0.064  0.071  -0.204  -0.063    0.072    1 15550
## r3   -0.082  0.073  -0.226  -0.081    0.060    1 15069
## r4   -0.083  0.072  -0.225  -0.083    0.057    1 14842
```

We can also extract each chain output to estimate the mode (which can be more accurate when the posterior distribution has a long tail).

```r
Nf1_output <- c(chain_output[[1]][,"Nf1"], chain_output[[2]][,"Nf1"],
                chain_output[[3]][,"Nf1"], chain_output[[4]][,"Nf1"])

Nf2_output <- c(chain_output[[1]][,"Nf2"], chain_output[[2]][,"Nf2"],
                chain_output[[3]][,"Nf2"], chain_output[[4]][,"Nf2"])

Nf3_output <- c(chain_output[[1]][,"Nf3"], chain_output[[2]][,"Nf3"],
                chain_output[[3]][,"Nf3"], chain_output[[4]][,"Nf3"])

Nf4_output <- c(chain_output[[1]][,"Nf4"], chain_output[[2]][,"Nf4"],
                chain_output[[3]][,"Nf4"], chain_output[[4]][,"Nf4"])

Nf1_mode <- round(density(Nf1_output)$x[which.max(density(Nf1_output)$y)], digits = 0)
Nf2_mode <- round(density(Nf2_output)$x[which.max(density(Nf2_output)$y)], digits = 0)
Nf3_mode <- round(density(Nf3_output)$x[which.max(density(Nf3_output)$y)], digits = 0)
Nf4_mode <- round(density(Nf4_output)$x[which.max(density(Nf4_output)$y)], digits = 0)
```

```
## [1] "Nf1 = 611"

## [1] "Nf2 = 123"

## [1] "Nf3 = 948"

## [1] "Nf4 = 889"
```