

Decreasing trends of chinstrap penguin breeding colonies in a region of major and ongoing environmental change: a statistical critique and reanalysis of Krüger (2023)

Analysis of sparse simulated population counts with GLMMs

Chris Oosthuizen, Murray Christian, Mzabalazo Ngwenya

2023-10-05

Contents

1 Purpose	1
2 Simulate data	2
3 GLMM fitting to ‘full’ data	7
4 Predicting population change: plot entire posterior distribution	10
5 Predicting population change between two time points	13
6 Predicting population change with sparse data	16
7 Predict from sparse data (multiple simulations)	20

1 Purpose

Sparse observations are the norm for chinstrap penguin populations, and this simulation study shows how sparse observations may influence estimates of population trends.

This script

1. simulates population counts for 26 sites over 60 years (the same simulation as in ‘PopulationTrend-SimulationStudy_GLMM comparison.rmd’, but with increased annual variation around the trend);
2. subset the data to 1980 to work with a shorter time-series;
3. randomly sample ‘observations’ from the ‘true’ population trend at every site to create a sparse ‘observed’ data set;
4. predict population trend from the sparse observation data using a GLMM.

- After doing this simulation and trend analysis once, we simulate 18 sparse data sets, analyse these, and plot the predictions.

Simulation studies, such as the one we conduct here, are useful to help understand the limitations of sparse observation data.

2 Simulate data

The following line of code sigma2.lambda <- 0.005 controls the annual variation in growth rate. If the value is small (e.g., sigma2.lambda <- 0.0005) the counts at each site increase or decrease relatively smoothly. Annual increase or decrease around the modeled trend increase as sigma2.lambda gets larger.

```
# load R packages
library(tidyverse)
library(MCMCglmm)
library(scales)
library(colorspace)
library(gridExtra)

# ! make sure library plyr / reshape2 (used in Krüger (2023) code) is not loaded !
unload_reshape2_if_loaded <- function() {
  # Check if the 'reshape2' package is loaded
  if ("package:reshape2" %in% search()) {
    # Unload the 'reshape2' package
    detach("package:reshape2", unload = TRUE)
    message("The 'reshape2' package has been unloaded.")
  } else {
    message("The 'reshape2' package is not loaded.")
  }
}

unload_reshape2_if_loaded()

#-----
# Simulate data
#-----

# Simulation based on Chapter 5 from State-Space Models for Population Counts from
# Bayesian Population Analysis using Winbugs by Marc Kery and Michael Schaub
# ISBN: 978-0-12-387020-9

# Make an empty list to save output in
list1 = list()  # for population counts
list2 = list()  # for lambda of each population

# Set seed for reproducibility
set.seed(12345)

# Choose how many populations and how many years you want to simulate
n.populations = 26 # Number of populations (max = 26)
n.years <- 60 # Number of years
start.year = 1960 # Start year
```

```

years = start.year:(start.year+n.years-1) # Year sequence

# simulate
for(i in 1:n.populations) {

  N1 <- runif(1, 500, 50000) # Initial population size
  mean.lambda <- runif(1, 0.92, 1.005) # Mean annual population growth rate
# sigma2.lambda <- 0.0005 # Process (temporal) variation of the growth rate
  sigma2.lambda <- 0.005 # Process (temporal) variation of the growth rate
  sigma2.y <- 0 # Variance of observation error (0 assumes 100% accurate counts)

  y <- N <- numeric(n.years)
  N[1] <- N1

  lambda <- rnorm(n.years-1, mean.lambda, sqrt(sigma2.lambda))

  for (t in 1:(n.years-1)){
    N[t+1] <- N[t] * lambda[t]
  }

  for (t in 1:n.years){
    y[t] <- rnorm(1, N[t], sqrt(sigma2.y))
  }

  # Save output in list for each iteration
  list1[[i]] = as.data.frame(y)
  list2[[i]] = as.data.frame(mean.lambda)
}

# Build data frame from simulations of count
df = bind_rows(list1)
names(df) = "count"
# add year to simulated counts
df$year = as.integer(rep(years,n.populations))
# add site to simulated counts
df$site = rep(LETTERS[1:n.populations], each = n.years)
# make df a tibble
df = as_tibble(df)

# Use list 2 (lambda) to generate a latitude value for each site that
# correlate with the site's growth rate (lambda)
lambda = bind_rows(list2)
lambda$site = LETTERS[1:n.populations]
lambda$noise = runif(n.populations, -1, 1)

df = merge(df, lambda, by = "site")

df$r = df$mean.lambda-1 # convert lambda to growth rate r
df$r100 = df$r * 100 # rescale

# create one latitude value per site where the mean latitude (-63 degrees S)
# increase or decrease based on the growth rate of the population plus a small
# random component

```

```

df = df %>%
  group_by(site) %>%
  mutate(latitude = -63 + r100 + noise) %>%
  ungroup()

# Inspect df: every site should have 1 unique latitude value
df %>%
  group_by(site) %>%
  summarise(count = n_distinct(latitude)) %>%
  summarise(max_sites_per_lat = max(count))

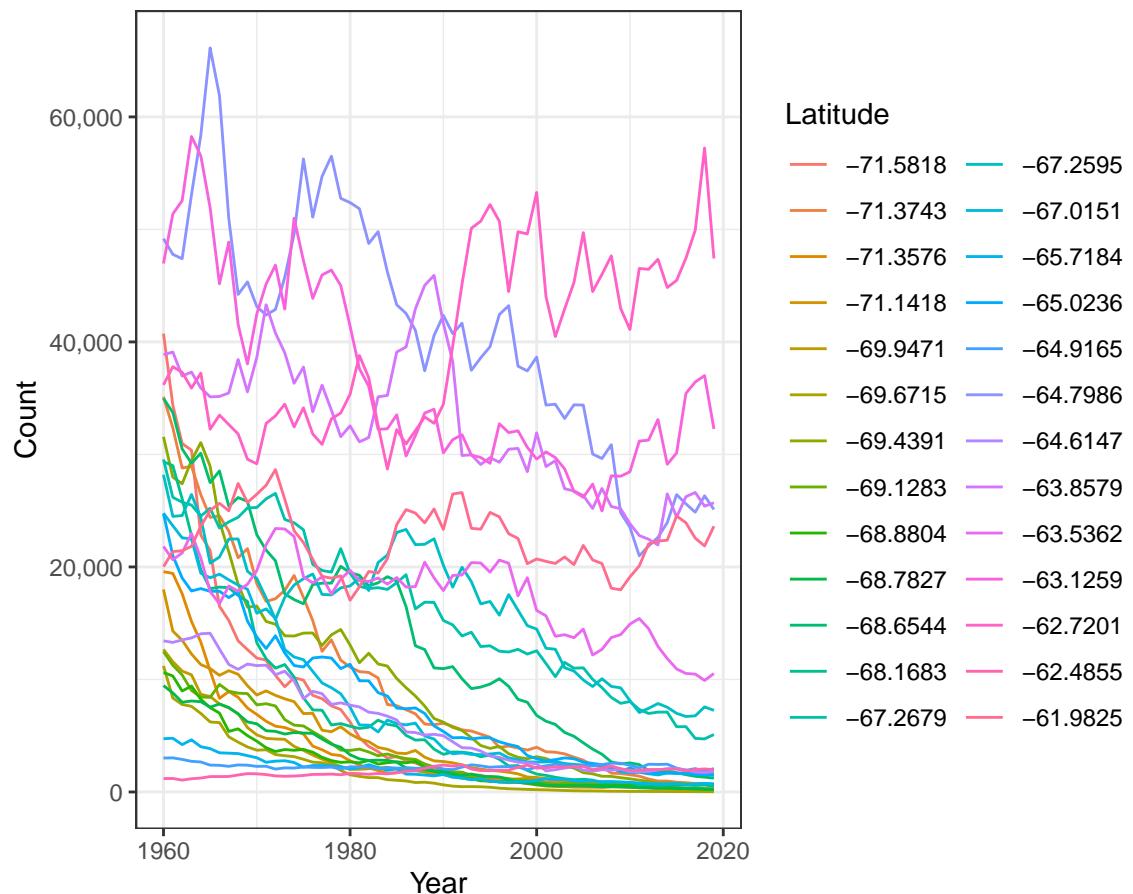
## # A tibble: 1 x 1
##   max_sites_per_lat
##   <int>
## 1 1

# Counts must be positive and integers
df = df %>%
  dplyr::filter(count > 0) %>%
  dplyr::select(site, count, year, latitude) %>%
  mutate_at(vars(latitude), round, 4) %>%
  mutate_at(vars(count), round, 0)

# Plot data used for fitting models: every sites has a count every year.
ggplot(df, aes(x = year, y = count, color = as.factor(latitude))) +
  geom_line() +
  labs(x = "Year", y = "Count") +
  scale_color_discrete(name = "Latitude") +
  theme_bw() +
  scale_y_continuous(label = comma) +
  labs(subtitle = "Simulated data (counts from 26 sites)")

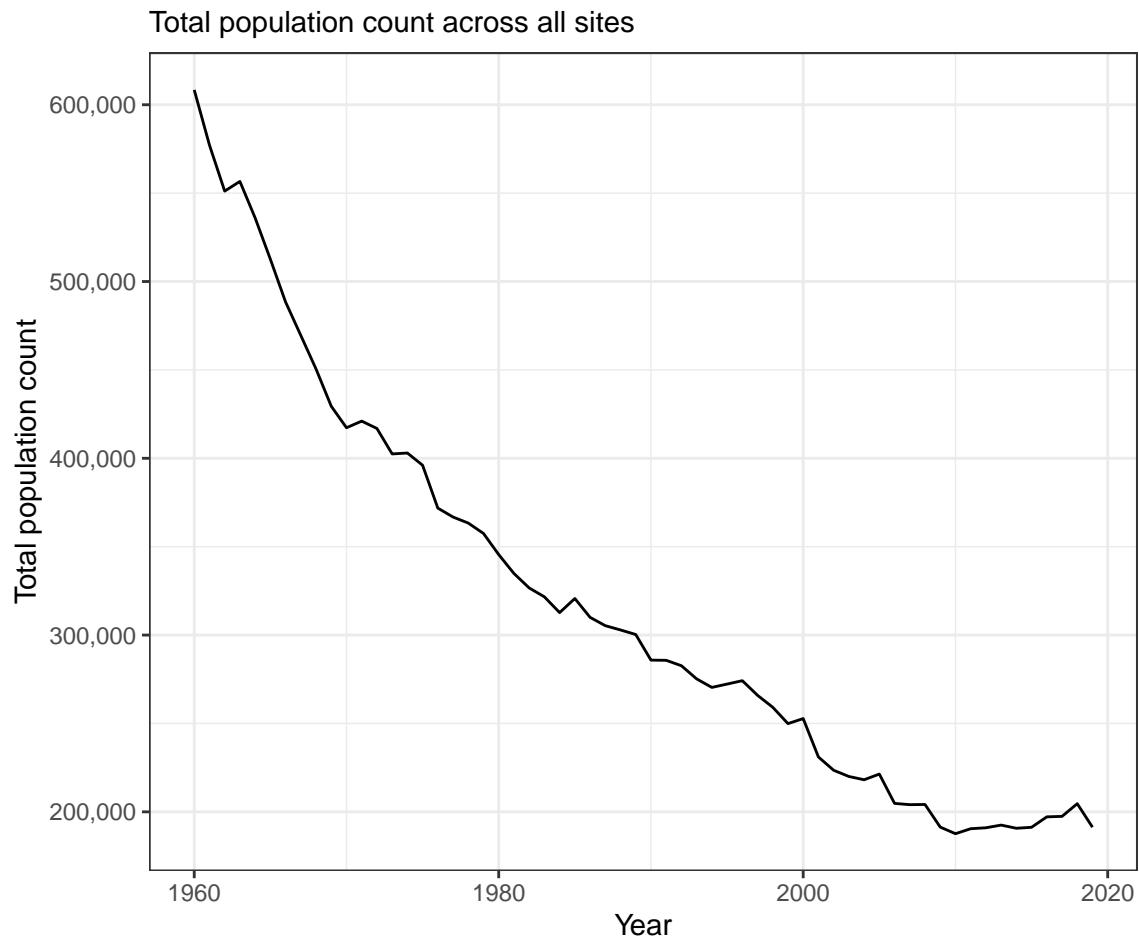
```

Simulated data (counts from 26 sites)



```
# How many penguins were there, per year, across all sites?
population = df %>%
  group_by(year) %>%
  summarise(totalN = sum(count))

# Plot total N (all sites) per year
ggplot(population, aes(x = year, y = totalN)) +
  geom_line() +
  labs(x = "Year", y = "Total population count") +
  theme_bw() +
  scale_y_continuous(label = comma) +
  labs(subtitle = "Total population count across all sites")
```



```

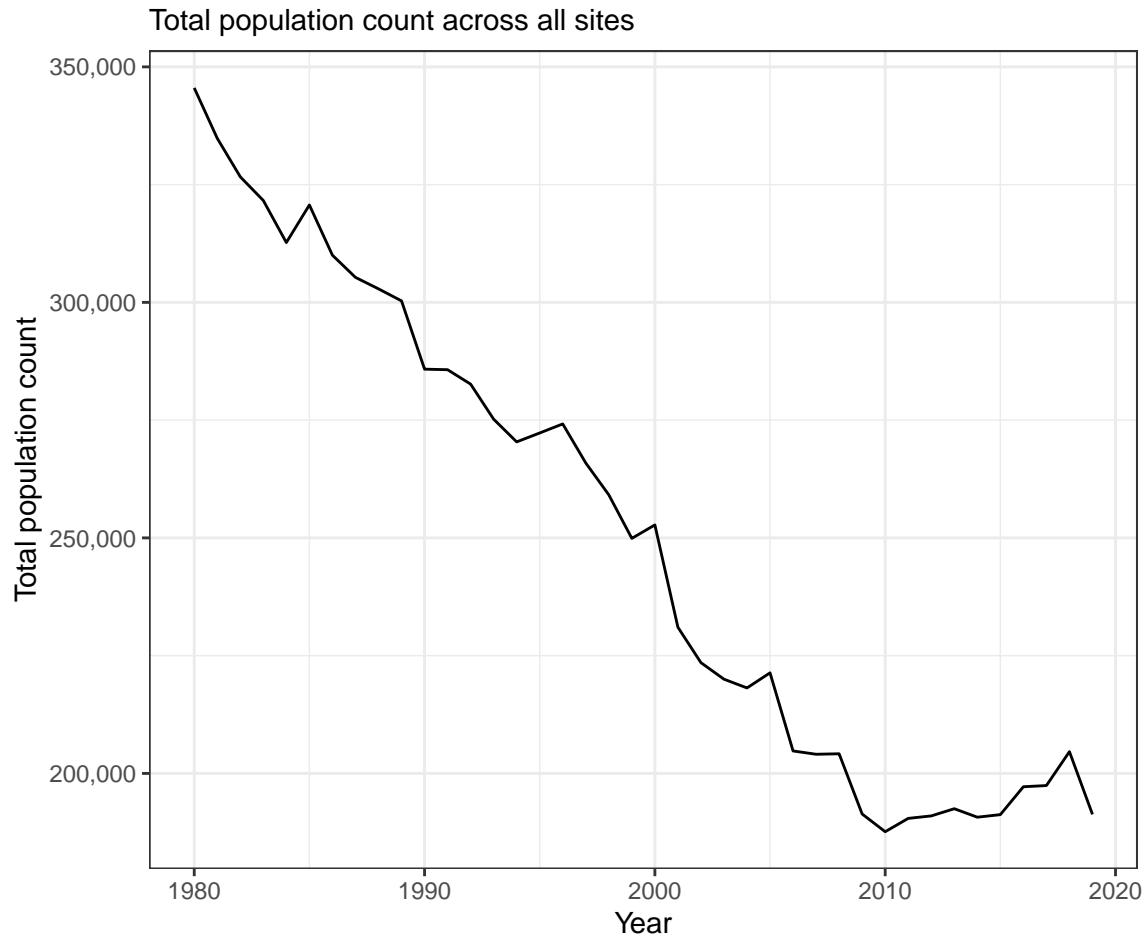
#--#
# Subset data to more recent period
#--#

df = df %>% dplyr::filter(year > 1979)

# How many penguins were there, per year, across all sites?
population = df %>%
  group_by(year) %>%
  summarise(totalN = sum(count))

# Plot total N (all sites) per year
ggplot(population, aes(x = year, y = totalN)) +
  geom_line() +
  labs(x = "Year", y = "Total population count") +
  theme_bw() +
  scale_y_continuous(label = comma) +
  labs(subtitle = "Total population count across all sites")

```



3 GLMM fitting to ‘full’ data

One observation per site every year.

```
# model convergence problems can be avoided by scaling variables
# to mean = 0, sd = 1
df$zyear = scale(df$year)
df$zlatitude = scale(df$latitude)

df = as.data.frame(df) # a data frame is expected by MCMCglmm

# Use prior from Kruger (2023) (change 133 to 26)
prior <- list(R = list(V = 1, nu = 0.002),
              G = list(G1 = list(V = diag(2), nu = 0.002,
                                  alpha.mu = rep(0, 2),
                                  alpha.V= diag(26, 2, 2))))
```



```
# Fit model with default prior or with prior specified above
mc2 <- MCMCglmm(count ~ zyear * zlatitude,
                  random = ~us(1 + zyear):site,
                  rcov=~units,
```

```

family="poisson",
data = df,
mev=NULL,start=NULL,
# prior=NULL,
prior=prior,
nodes="ALL", scale=TRUE,
nitt=13000, thin=10, burnin=3000, pr=T,
pl=FALSE, verbose=TRUE, DIC=TRUE, singular.ok=FALSE, saveX=TRUE,
saveZ=TRUE, saveXL=TRUE, slice=FALSE, ginverse=NULL, trunc=FALSE)

# do a prediction with random effects:
pred <- data.frame(predict(mc2,
                           newdata=df,
                           type="response",
                           marginal=NULL,
                           interval="prediction",
                           posterior="all"))

df$fit.mc2 <- pred$fit
df$lwr.mc2 <- pred$lwr
df$upr.mc2 <- pred$upr

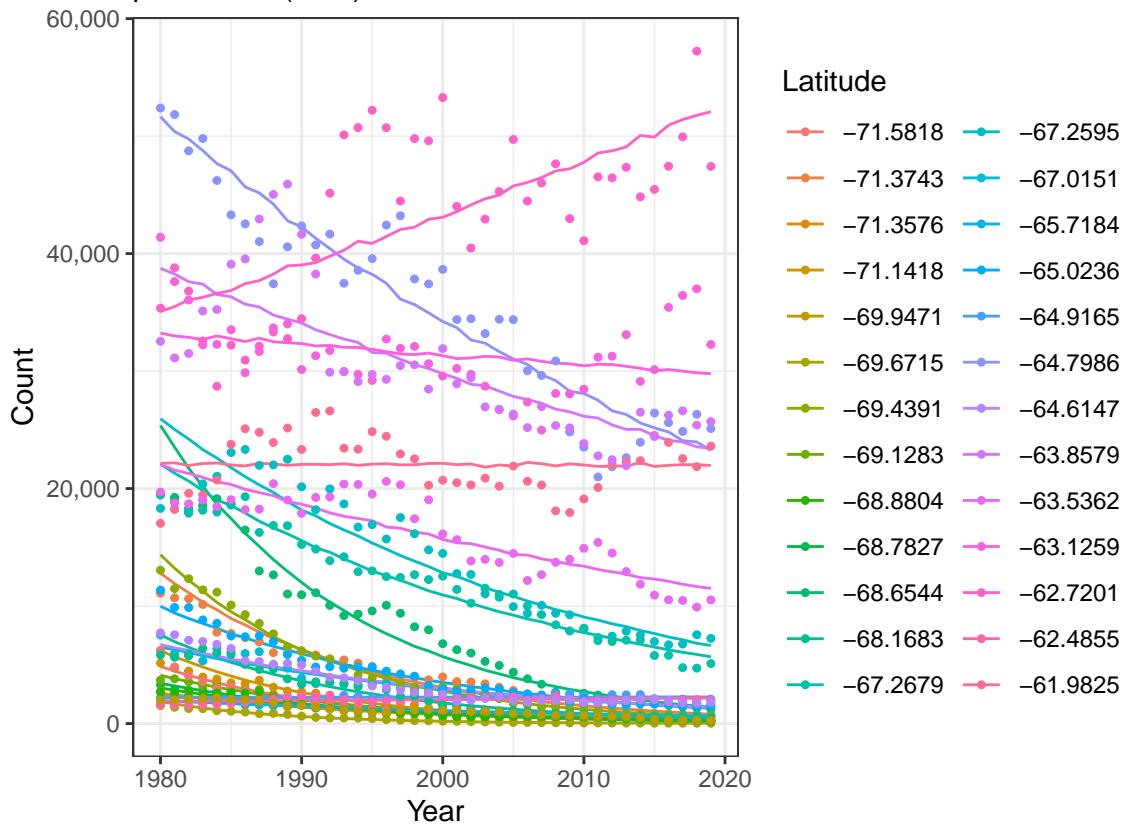
# predict with observed data
fit.mc2p = ggplot(df, aes(x = year, y = fit.mc2, color = as.factor(latitude))) +
  geom_line() + theme_bw() +
  labs(x = "Year", y = "Count") +
  scale_color_discrete(name = "Latitude") +
  scale_y_continuous(label = comma) +
  # add observed data
  geom_point(aes(x = year, y = count, color = as.factor(latitude)), size = 0.9) +
  labs(title = "MCMCglmm: prediction with random effects",
       subtitle = "Observed data (points) match\npredictions (lines)")

fit.mc2p

```

MCMCglmm: prediction with random effects

Observed data (points) match
predictions (lines)

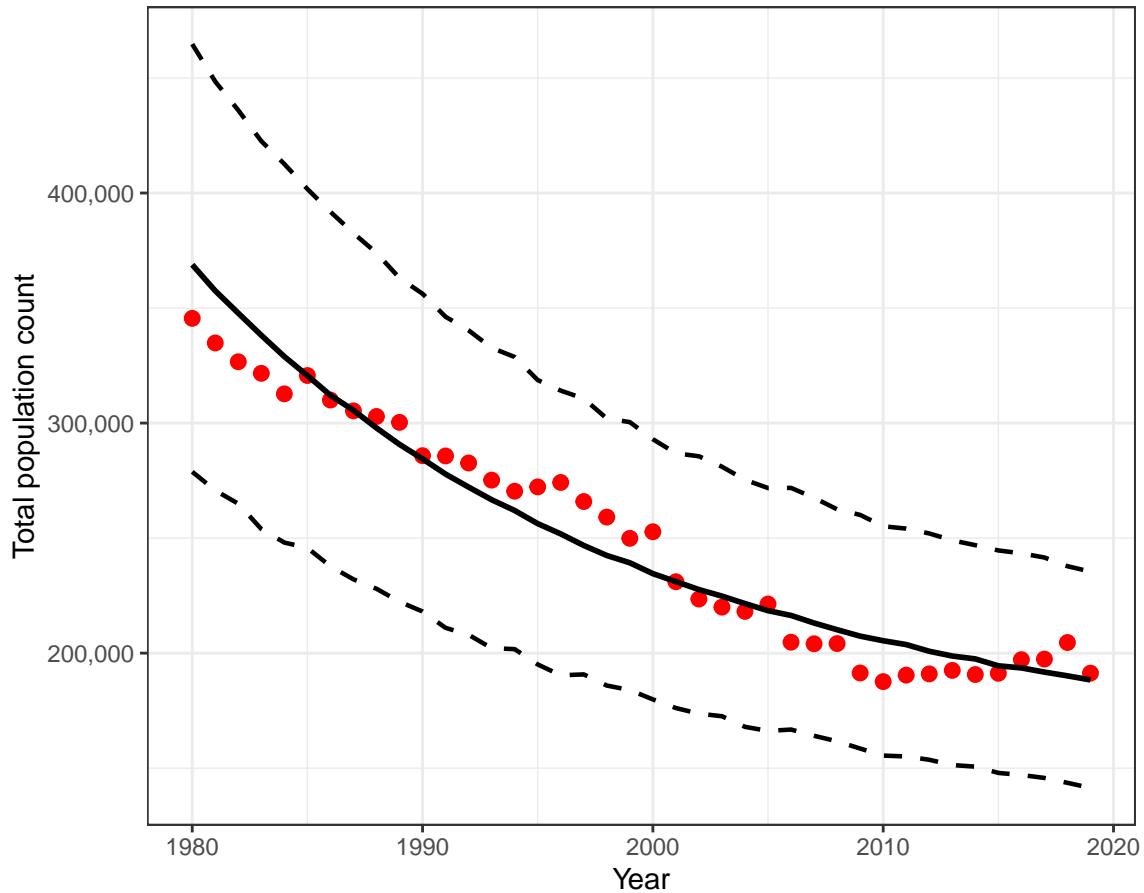


```
# calculate total population size predicted (total N across all sites)
pop_predict_mc2 = df %>%
  group_by(year) %>%
  summarise(total_pred = sum(fit.mc2),
            min_pred = sum(lwr.mc2),
            max_pred = sum(upr.mc2))

# Plot total N observed per year and the predicted total abundance per year
pop_predict_mc2p = ggplot() +
  geom_point(data = population, aes(x = year, y = totalN,
                                    color = "Observed (simulated) data"), color = "red", size = 2.3) +
  geom_line(data = pop_predict_mc2, aes(x = year, y = total_pred,
                                         color = "Model predicted count"), color = "black", lty = 1, linewidth = 1) +
  geom_line(data = pop_predict_mc2, aes(x = year, y = min_pred,
                                         color = "Model predicted count"), color = "black", lty = 2, linewidth = 0.8) +
  geom_line(data = pop_predict_mc2, aes(x = year, y = max_pred,
                                         color = "Model predicted count"), color = "black", lty = 2, linewidth = 0.8) +
  labs(x = "Year", y = "Total population count") +
  theme_bw() +
  scale_y_continuous(label = comma) +
  labs(subtitle = "Total observed and predicted counts across 26 sites: sites counted every year") +
  guides(color=guide_legend(title="Data source")) +
  theme(legend.position = c(0.7, 0.8))
```

```
pop_predict_mc2p
```

Total observed and predicted counts across 26 sites: sites counted every year



4 Predicting population change: plot entire posterior distribution

```
# extract posterior draws of fixed effects and random effects
posterior <- as.matrix(mc2$Sol)

# collect site-level information
site_and_lat <- df %>%
  as_tibble() %>%
  select(site, zlatitude) %>%
  distinct()

# define function to predict population size every year, with or without random effects
get_annual_predictions <- function(posterior,
                                    site_and_lat,
                                    all_years,
                                    use_random_effects = FALSE) {
```

```

# matrices for predictions at each site in year t
# each row is a prediction from a different posterior sample, each column is a site
pred_pop_per_site_all <- matrix(NA, nrow = nrow(posterior), ncol = nrow(site_and_lat))

for (s in 1:nrow(posterior)) {
  theta <- posterior[s,]
  for (j in 1:nrow(site_and_lat)) {
    site <- site_and_lat$site[j]
    zlatitude <- site_and_lat$zlatitude[j]

    # predict pop at site j in first year
    lin_pred <- theta[["(Intercept)"]] +
      theta[["zyear"]] * all_years +
      theta[["zlatitude"]] * zlatitude +
      theta[["zyear:zlatitude"]] * all_years * zlatitude
    if (use_random_effects) {
      lin_pred <- lin_pred +
        theta[ str_c("(Intercept).site.",site) ] +
        theta[ str_c("zyear.site.",site) ] * all_years
    }
    pred_pop_per_site_all[s,j] <- exp( lin_pred )
  }
}

# sum over sites for population level predictions
pred_pop_all <- rowSums(pred_pop_per_site_all)

# outputs
predictions <- list(pop_per_site_all = pred_pop_per_site_all,
                     pop_all = pred_pop_all)
predictions
}

this_year = unique(df$year)

# Initialize a data frame to store results
all_years_df <- data.frame()

#this_year = c(1960)

for(t in this_year) {

  all_years <- (t - mean(df$year)) / sd(df$year)

  # Make predictions with and without random effects
  pred_re <- get_annual_predictions(posterior, site_and_lat, all_years,
                                      use_random_effects = TRUE)

  # Save the individual list into the outer list
  individual_result <- data.frame(pred_re$pop_all)
  individual_result$year = t
}

```

```

# Append the individual result to the data frame
all_years_df <- rbind(all_years_df, individual_result)

}

head(all_years_df)

##   pred_re.pop_all year
## 1      368714.8 1980
## 2      368088.8 1980
## 3      359112.7 1980
## 4      364221.1 1980
## 5      371989.7 1980
## 6      365583.5 1980

tail(all_years_df)

##           pred_re.pop_all year
## 39995      187169.3 2019
## 39996      182801.7 2019
## 39997      189512.8 2019
## 39998      184272.6 2019
## 39999      190894.1 2019
## 40000      187586.8 2019

# Plot total N observed per year and the predicted total abundance per year

pop_predict_mc2p_all =
  ggplot() +
  # posterior samples
  geom_jitter(data = all_years_df,
              aes(x = as.numeric(year), y = pred_re.pop_all),
              color = "darkblue", size = 0.5, height = 0, width = 0.25,
              alpha = 0.2, stroke = NA) +
  geom_point(data = population, aes(x = year, y = totalN,
                                    color = "Observed (simulated) data"), color = "red", size = 2.3) +
  geom_line(data = pop_predict_mc2, aes(x = year, y = total_pred,
                                         color = "Model predicted count"), color = "black", lty = 1, size = 1) +
  geom_line(data = pop_predict_mc2, aes(x = year, y = min_pred,
                                         color = "Model predicted count"), color = "black", lty = 2, linewidth = 1) +
  geom_line(data = pop_predict_mc2, aes(x = year, y = max_pred,
                                         color = "Model predicted count"), color = "black", lty = 2, linewidth = 1) +
  labs(x = "Year", y = "Total population count") +
  theme_bw() +
  scale_y_continuous(label = comma) +
  labs(subtitle = "Total observed and predicted counts across 26 sites: sites counted every year") +
  guides(color=guide_legend(title="Data source")) +
  theme(legend.position = c(0.7, 0.8))

## Warning: Using 'size' aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use 'linewidth' instead.

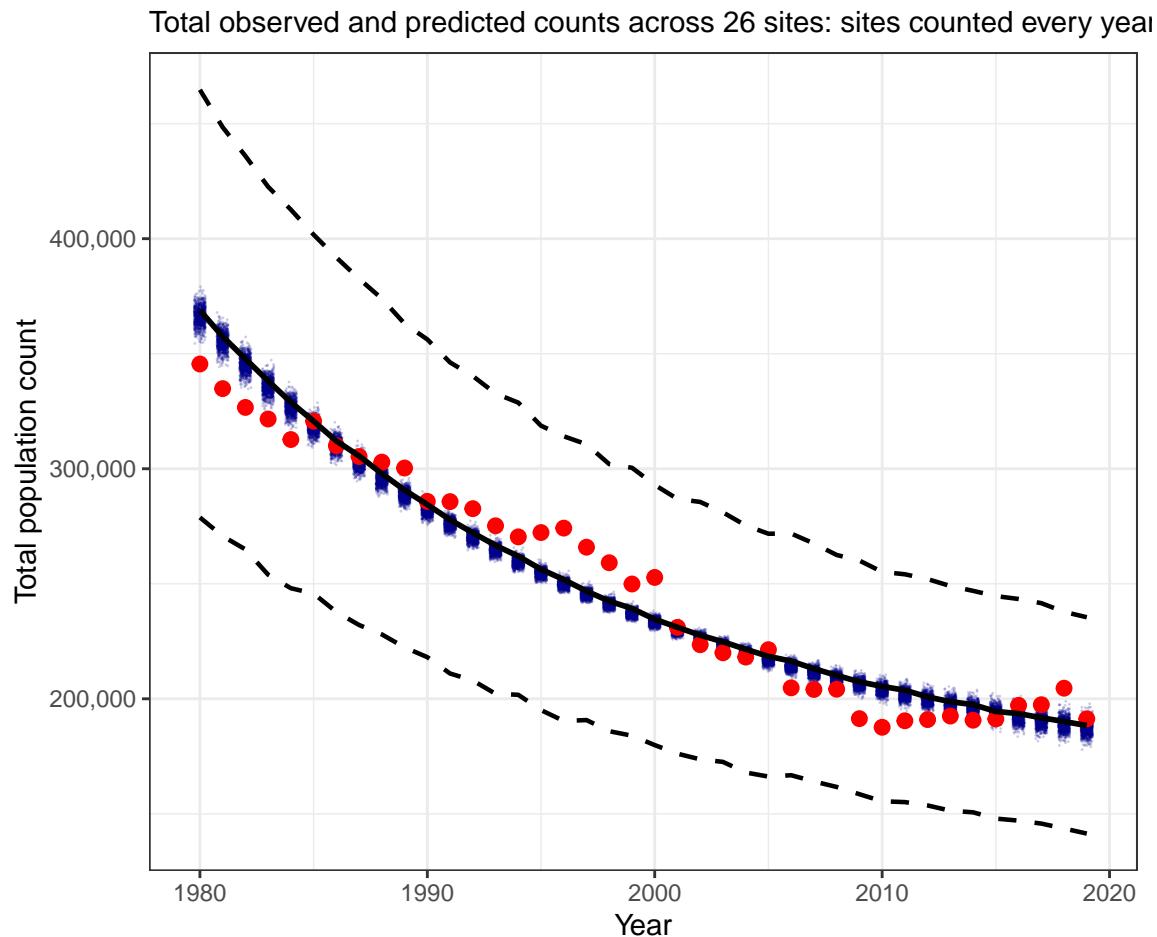
```

```

## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

```

```
pop_predict_mc2p_all
```



5 Predicting population change between two time points

```

# map years which to predict to (standardised scale)
# Here, the first year is 1980 and the last year is 2019 (40 year change)

year1 = 1980
year2 = 2019

first_year <- (year1 - mean(df$year)) / sd(df$year)
last_year <- (year2 - mean(df$year)) / sd(df$year)

# define function to predict with or without random effects
get_predictions <- function(posterior,
                           site_and_lat,

```

```

        first_year,
        last_year,
        use_random_effects = FALSE) {
# matrices for predictions at each site in year 1 (1980) and year 2 (2019)
# each row is a prediction from a different posterior sample, each column is a site
pred_pop_per_site.first <- matrix(NA, nrow = nrow(posterior), ncol = nrow(site_and_lat))
pred_pop_per_site.last <- matrix(NA, nrow = nrow(posterior), ncol = nrow(site_and_lat))

for (s in 1:nrow(posterior)) {
  theta <- posterior[s,]
  for (j in 1:nrow(site_and_lat)) {
    site <- site_and_lat$site[j]
    zlatitude <- site_and_lat$zlatitude[j]

    # predict pop at site j in first year
    lin_pred <- theta["(Intercept)"] +
      theta["zyear"] * first_year +
      theta["zlatitude"] * zlatitude +
      theta["zyear:zlatitude"] * first_year * zlatitude
    if (use_random_effects) {
      lin_pred <- lin_pred +
        theta[ str_c("(Intercept).site.",site) ] +
        theta[ str_c("zyear.site.",site) ] * first_year
    }
    pred_pop_per_site.first[s,j] <- exp( lin_pred )

    # predict pop at site j in last year
    lin_pred <- theta["(Intercept)"] +
      theta["zyear"] * last_year +
      theta["zlatitude"] * zlatitude +
      theta["zyear:zlatitude"] * last_year * zlatitude
    if (use_random_effects) {
      lin_pred <- lin_pred +
        theta[ str_c("(Intercept).site.",site) ] +
        theta[ str_c("zyear.site.",site) ] * last_year
    }
    pred_pop_per_site.last[s,j] <- exp( lin_pred )
  }
}

# sum over sites for population level predictions
pred_pop.first <- rowSums(pred_pop_per_site.first)
pred_pop.last <- rowSums(pred_pop_per_site.last)

# percent change from year1 to year2
pred_pop_change <- 100 * ( pred_pop.last - pred_pop.first ) / pred_pop.first

# outputs
predictions <- list(pop_per_site.first = pred_pop_per_site.first,
                      pop_per_site.last = pred_pop_per_site.last,
                      pop.first = pred_pop.first,
                      pop.last = pred_pop.last,
                      pop_change = pred_pop_change)

```

```

    predictions
}

# Make predictions with and without random effects
pred_re <- get_predictions(posterior, site_and_lat, first_year, last_year,
                           use_random_effects = TRUE)

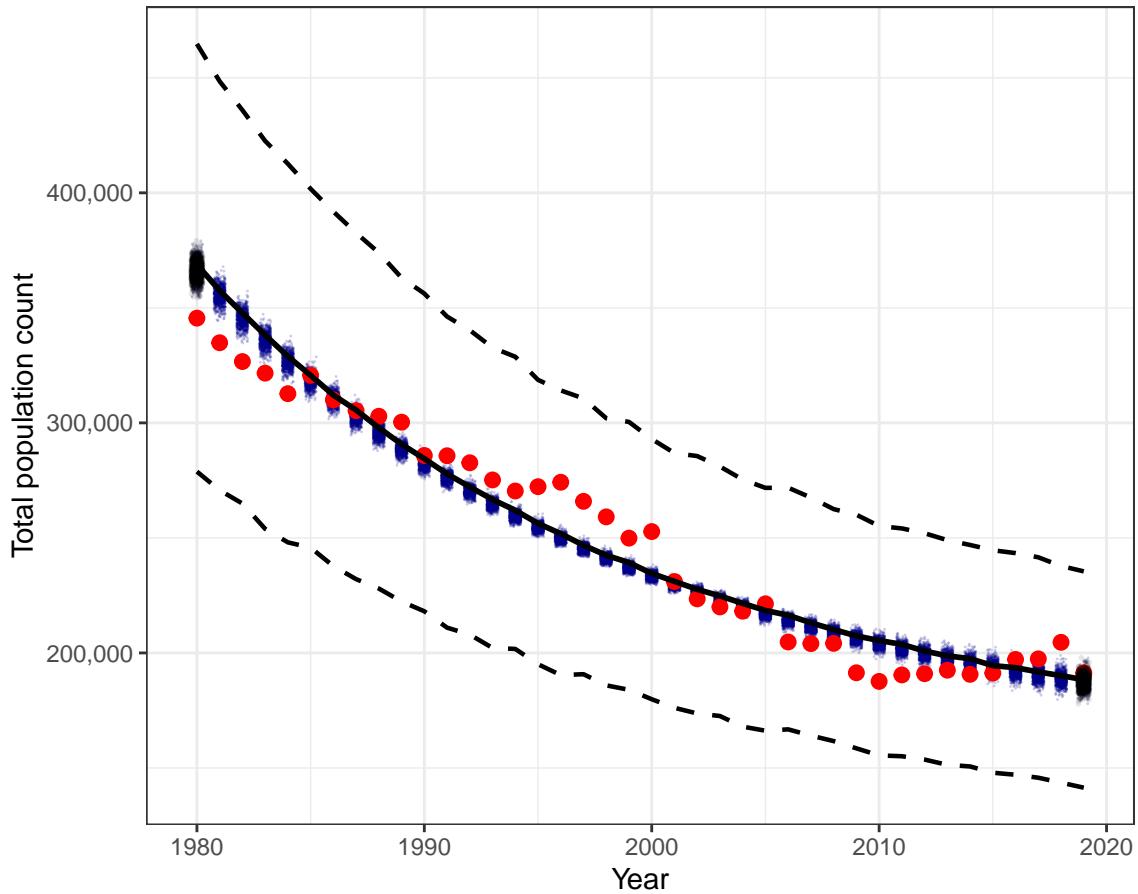
# estimated population size in year1 (with random effects)
pred_first = as.data.frame(pred_re$pop.first)
names(pred_first) = "pred_first"

# estimated population size in year2 (with random effects)
pred_last = as.data.frame(pred_re$pop.last)
names(pred_last) = "pred_last"

pop_predict_mc2p_all +
  geom_jitter(data = pred_first, aes(x = year1, y = pred_first),
              col = "black", size = 1, height = 0, width = 0.25,
              alpha = 0.1, stroke = NA) +
  geom_jitter(data = pred_last, aes(x = year2, y = pred_last),
              col = "black", size = 1, height = 0, width = 0.25,
              alpha = 0.1, stroke = NA)

```

Total observed and predicted counts across 26 sites: sites counted every year



6 Predicting population change with sparse data

The above prediction was made with a observed count every year. The MAPPPD data, in contrast, are sparse and unbalanced. Here, we sub-sample counts from the time series and compare the predicted population trend to the true population trend.

```
# size of 'full' data frame
dim(df)

## [1] 1040     9

# clean up
df_sparse = df %>% dplyr::select(site, count, year, latitude)

# assign a random 'weight' to every site so that some sites are sampled more frequently
# and others are sampled less frequently.
set.seed (12345)
characters = LETTERS[c(1:26)]
wghts = data.frame(site = characters,
                   weights = floor(runif(length(unique(characters)),
                                         min=10, max=100)))
```

```

# add weights to df
df_sparse = merge(df_sparse, wghts, by = "site")
#head(df_sparse)

set.seed (1234)
sparse_counts = df_sparse %>%
  dplyr::slice_sample(prop = 0.1, replace = FALSE, weight_by = weights)

# Check if the value is less than 26 sites.
# while (length(unique(sparse_counts$site)) < 26) {
# while (min(table(sparse_counts$site)) < 2) {
while (length(unique(sparse_counts$site)) < 26 || 
  min(table(sparse_counts$site)) < 2) {

  # If the value is less than 26, sample again
sparse_counts = df_sparse %>%
  dplyr::slice_sample(prop = 0.1, replace = FALSE, weight_by = weights)

}

# make sure there are 26 sites with at least 2 counts each in the sparse data set.
table(sparse_counts$site)

```

```

## 
## A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
## 4 2 9 2 6 2 2 6 7 4 3 4 7 2 4 3 2 3 2 5 4 5 5 5 2 4

```

```
length(unique(sparse_counts$site))
```

```
## [1] 26
```

```
min(table(sparse_counts$site))
```

```
## [1] 2
```

```

sparse_counts = sparse_counts %>%
  group_by(site) %>%
  mutate(Ncount = length(count)) %>%
  ungroup()

```

```

sparse_counts_N = sparse_counts %>%
  group_by(site) %>%
  summarise(Ncount = mean(Ncount))

```

```
summary(sparse_counts_N$Ncount)
```

```

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##          2         2       4        4       5        9

```

```

#NB! You have to recalculate the standardizations to mean = 0, sd = 1!
#head(sparse_counts)
sparse_counts$zyear = scale(sparse_counts$year)
sparse_counts$zlatitude = scale(sparse_counts$latitude)

# must be a data frame for MCMCglmm
sparse_counts = as.data.frame(sparse_counts)

# Fit model m2 from above using SPARSE DATA
mc_sparse <- MCMCglmm(count ~ zyear * zlatitude,
                        random = ~us(1 + zyear):site,
                        rcov=~units,
                        family="poisson",
                        data = sparse_counts,
                        mev=NULL, start=NULL,
                        # prior=NULL,
                        prior=prior,
                        nodes="ALL", scale=TRUE,
                        nitt=13000, thin=10, burnin=3000, pr=T,
                        pl=FALSE, verbose=TRUE, DIC=TRUE, singular.ok=FALSE, saveX=TRUE,
                        saveZ=TRUE, saveXL=TRUE, slice=FALSE, ginverse=NULL, trunc=FALSE)

# construct an hypothetical dataframe to predict to
# need to predict to z-standardized variables
Z1 = dplyr::select(sparse_counts, year, latitude)
Z2 <- scale(Z1)
attr(Z2, "scaled:center")

##      year   latitude
## 2001.35577 -66.78566

attr(Z2, "scaled:scale")

##      year   latitude
## 11.048805  3.089907

ave_ss = attr(Z2, "scaled:center")[[1]]
ave_lat = attr(Z2, "scaled:center")[[2]]

sd_ss = attr(Z2, "scaled:scale")[[1]]
sd_lat = attr(Z2, "scaled:scale")[[2]]

years = data.frame(year=c(1980:2019)) # extrapolate to 1960

# collect site-level information
pops = sparse_counts %>%
  select(site, latitude) %>%
  distinct()

popy<-merge(pops,years)
popy$count<-c(0) ### MCMCglmm needs a column with the response variable

```

```

popy$zyear = (popy$year - ave_ss)/sd_ss
popy$zlatitude = (popy$latitude - ave_lat)/sd_lat
#head(popy)
length(unique(popy$site))

## [1] 26

pred_sparse <- data.frame(predict(mc_sparse,
                                    newdata = popy,
                                    type="response",
                                    marginal=NULL,           # crucial, and not default code.
                                    interval="prediction",
                                    posterior="all"))

#head(pred_sparse)

popy$Zfit_sparse = pred_sparse$fit
popy$Zlwr_sparse = pred_sparse$lwr
popy$Zupr_sparse = pred_sparse$upr

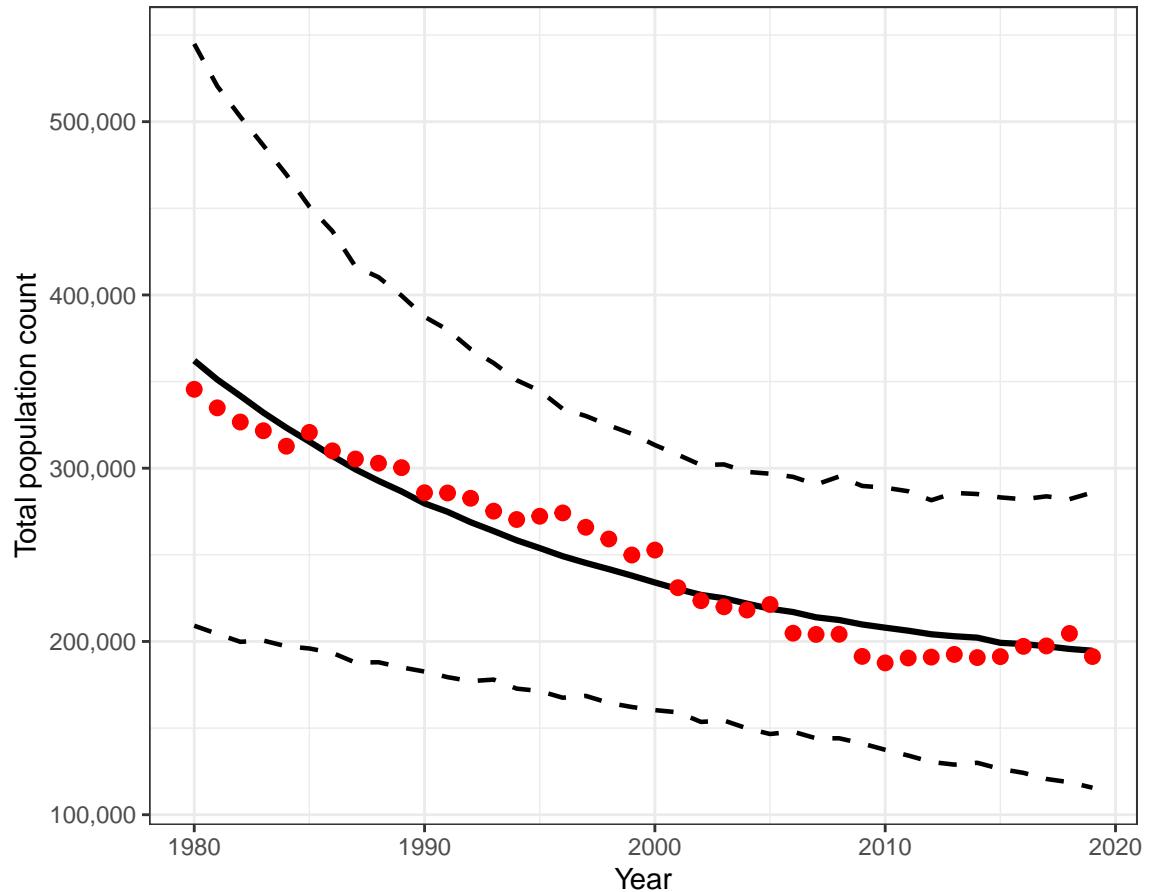
pop_predict_sparse = popy %>%
  dplyr::group_by(year) %>%
  dplyr::summarise(total_pred = sum(Zfit_sparse),
                  min_pred = sum(Zlwr_sparse),
                  max_pred = sum(Zupr_sparse))

# plot
pop_predict.p = ggplot(data = pop_predict_sparse) +
  geom_line(aes(x = year, y = total_pred),
            lty = 1, linewidth = 1.1)+
  geom_line(aes(x = year, y = min_pred, color = "Model predicted count"),
            color = "black", lty = 2, linewidth = 0.8)+
  geom_line(aes(x = year, y = max_pred, color = "Model predicted count"),
            color = "black", lty = 2, linewidth = 0.8)+
  labs(x = "Year", y = "Total population count") +
  theme_bw()+
  scale_y_continuous(label = comma)+
  scale_x_continuous(breaks = seq(1980, 2020, by = 10),
                     labels = seq(1980, 2020, by = 10))+ 
  labs(subtitle = "Total predicted counts across all sites: sparse data")+
  guides(color=guide_legend(title="Data source"))+
  theme(legend.position = c(0.7, 0.9)) +
  geom_point(data = population, aes(x = year, y = totalN,
                                    color = "Observed (simulated) data"), color = "red", size = 2.3)

pop_predict.p # Decent prediction; uncertainty slightly higher than 'annual count model'

```

Total predicted counts across all sites: sparse data



7 Predict from sparse data (multiple simulations)

Above was one example of a sparse data set. Now, simulate 18 sparse data sets, analyse these, and plot the predictions.

```
# 1. Sample a random number of 'observations'
# 2. Fit a GLMM to those observations
# 3. Predict population trend
# 4. Compare to 'true' overall population size

list_sparse = list()
plots_list = list()
posterior_list = list()

# simulate
for(i in 1:18) {

  # n = sample(80:170, 1)
  my_seq <- seq(0.1, 0.2, by = 0.01)
  # Sample one element from the sequence
  n = sample(my_seq, 1)
```

```

sparse_counts = df_sparse %>%
  dplyr::slice_sample(prop = n, replace = FALSE, weight_by = weights)

# Check if the value is less than 26 sites.
# while (length(unique(sparse_counts$site)) < 26) {
#   while (min(table(sparse_counts$site)) < 2) {
while (length(unique(sparse_counts$site)) < 26 || 
  min(table(sparse_counts$site)) < 2) {

  # If the value is less than 26, sample again
  sparse_counts = df_sparse %>%
    dplyr::slice_sample(prop = n, replace = FALSE, weight_by = weights)

}

length(unique(sparse_counts$site))
table(sparse_counts$site)

# Standardize year and latitude for each individual random data set
sparse_counts$zyear = scale(sparse_counts$year)
sparse_counts$zlatitude = scale(sparse_counts$latitude)

# Fit model m2 from above using SPARSE DATA
mc_sparse <- MCMCglmm(count ~ zyear * zlatitude,
                        random = ~us(1 + zyear):site,
                        rcov=~units,
                        family="poisson",
                        data = sparse_counts,
                        mev=NULL,start=NULL,
                        # prior=NULL,
                        prior=prior,
                        nodes="ALL", scale=TRUE,
                        nitt=13000, thin=10, burnin=3000, pr=T,
                        pl=FALSE, verbose=TRUE, DIC=TRUE, singular.ok=FALSE, saveX=TRUE,
                        saveZ=TRUE, saveXL=TRUE, slice=FALSE, ginverse=NULL, trunc=FALSE)

# construct an hypothetical dataframe to predict to
# need to predict to z-standardized variables
Z1 = dplyr::select(sparse_counts, year, latitude)
Z2 <- scale(Z1)
attr(Z2,"scaled:center")
attr(Z2,"scaled:scale")

ave_ss = attr(Z2,"scaled:center")[[1]]
ave_lat = attr(Z2,"scaled:center")[[2]]

sd_ss = attr(Z2,"scaled:scale")[[1]]
sd_lat = attr(Z2,"scaled:scale")[[2]]

years = data.frame(year=c(1980:2019)) # extrapolate to 1960

# collect site-level information
pops = sparse_counts %>%

```

```

  select(site, latitude ) %>%
  distinct()

popy<-merge(pops,years)
popy$count<-c(0) ### MCMCglmm needs a column with the response variable

popy$zyear = (popy$year - ave_ss)/sd_ss
popy$zlatitude = (popy$latitude - ave_lat)/sd_lat

# head(popy)

length(unique(popy$site))

pred_sparse <- data.frame(predict(mc_sparse,
                                    newdata = popy,
                                    type="response",
                                    marginal=NULL,      # crucial, and not default code.
                                    interval="prediction",
                                    posterior="all"))

#head(pred_sparse)

popy$Zfit_sparse = pred_sparse$fit
popy$Zlwr_sparse = pred_sparse$lwr
popy$Zupr_sparse = pred_sparse$upr

pop_predict_sparse = popy %>%
  dplyr::group_by(year) %>%
  dplyr::summarise(total_pred = sum(Zfit_sparse),
                  min_pred = sum(Zlwr_sparse),
                  max_pred = sum(Zupr_sparse),
                  iteration = mean(i),
                  observation.samplesize = n)

# Save output in list for each iteration
list_sparse[[i]] = pop_predict_sparse

heat = ggplot(sparse_counts, aes(x = as.numeric(year),
                                   y = site,fill = weights/100)) +
  geom_tile() +
  scale_fill_continuous_sequential(palette = "BluGrn", rev = F) +
  guides(fill = FALSE) +
  theme_bw() +
  ylab("Site") +
  xlab("Year") +
  theme(axis.text.x=element_text(size=8),
        axis.title.x=element_text(size=10),
        axis.text.y = element_text(size = 6),
        axis.title.y=element_text(size=10),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  scale_x_continuous(breaks = seq(1980, 2020, by = 10)) +
  scale_y_discrete(limits=rev) +

```

```

  labs(subtitle = paste("Data", i, "-", length(unique(sparse_counts$site)),
                        "sites, min sample = ", min(table(sparse_counts$site)))))

# Add the plot to the list
plots_list[[i]] <- heat

#-----
# extract posterior draws of fixed effects and random effects
posterior <- as.matrix(mc_sparse$Sol)

# collect site-level information
site_and_lat <- sparse_counts %>%
  as_tibble() %>%
  select(site, zlatitude) %>%
  distinct()

site_and_lat

# map years which to predict to (standardised scale)
# Here, the first year is 1980 and the last year is 2019 (40 year change)

year1 = 1980
year2 = 2019

first_year <- (year1 - mean(df$year)) / sd(df$year)
last_year <- (year2 - mean(df$year)) / sd(df$year)

# define function to predict with or without random effects
get_predictions <- function(posterior,
                             site_and_lat,
                             first_year,
                             last_year,
                             use_random_effects = FALSE) {
  # matrices for predictions at each site in year 1 (1980) and year 2 (2019)
  # each row is a prediction from a different posterior sample, each column is a site
  pred_pop_per_site.first <- matrix(NA, nrow = nrow(posterior),
                                      ncol = nrow(site_and_lat))
  pred_pop_per_site.last <- matrix(NA, nrow = nrow(posterior),
                                    ncol = nrow(site_and_lat))

  for (s in 1:nrow(posterior)) {
    theta <- posterior[s,]
    for (j in 1:nrow(site_and_lat)) {
      site <- site_and_lat$site[j]
      zlatitude <- site_and_lat$zlatitude[j]

      # predict pop at site j in first year
      lin_pred <- theta["(Intercept)"] +
        theta["zyear"] * first_year +
        theta["zlatitude"] * zlatitude +
        theta["zyear:zlatitude"] * first_year * zlatitude
      if (use_random_effects) {
        lin_pred <- lin_pred +

```

```

        theta[ str_c("(Intercept).site.",site) ] +
        theta[ str_c("zyear.site.",site) ] * first_year
    }
    pred_pop_per_site.first[s,j] <- exp( lin_pred )

    # predict pop at site j in last year
    lin_pred <- theta["(Intercept)"] +
        theta["zyear"] * last_year +
        theta["zlatitude"] * zlatitude +
        theta["zyear:zlatitude"] * last_year * zlatitude
    if (use_random_effects) {
        lin_pred <- lin_pred +
            theta[ str_c("(Intercept).site.",site) ] +
            theta[ str_c("zyear.site.",site) ] * last_year
    }
    pred_pop_per_site.last[s,j] <- exp( lin_pred )
}
}

# sum over sites for population level predictions
pred_pop.first <- rowSums(pred_pop_per_site.first)
pred_pop.last <- rowSums(pred_pop_per_site.last)

# percent change from year1 to year2
pred_pop_change <- 100 * ( pred_pop.last - pred_pop.first ) / pred_pop.first

# outputs
predictions <- list(pop_per_site.first = pred_pop_per_site.first,
                      pop_per_site.last = pred_pop_per_site.last,
                      pop.first = pred_pop.first,
                      pop.last = pred_pop.last,
                      pop_change = pred_pop_change)
predictions
}

# Make predictions with and without random effects
pred_re <- get_predictions(posterior, site_and_lat, first_year, last_year,
                           use_random_effects = TRUE)

# estimated population size in year1 (with random effects)
pred_first = as.data.frame(pred_re$pop.first)
names(pred_first) = "pred_first"

# estimated population size in year2 (with random effects)
pred_last = as.data.frame(pred_re$pop.last)
names(pred_last) = "pred_last"

iteration = i

prediction_sum = data.frame(pred_first, pred_last, iteration)

posterior_list[[i]] = prediction_sum

```

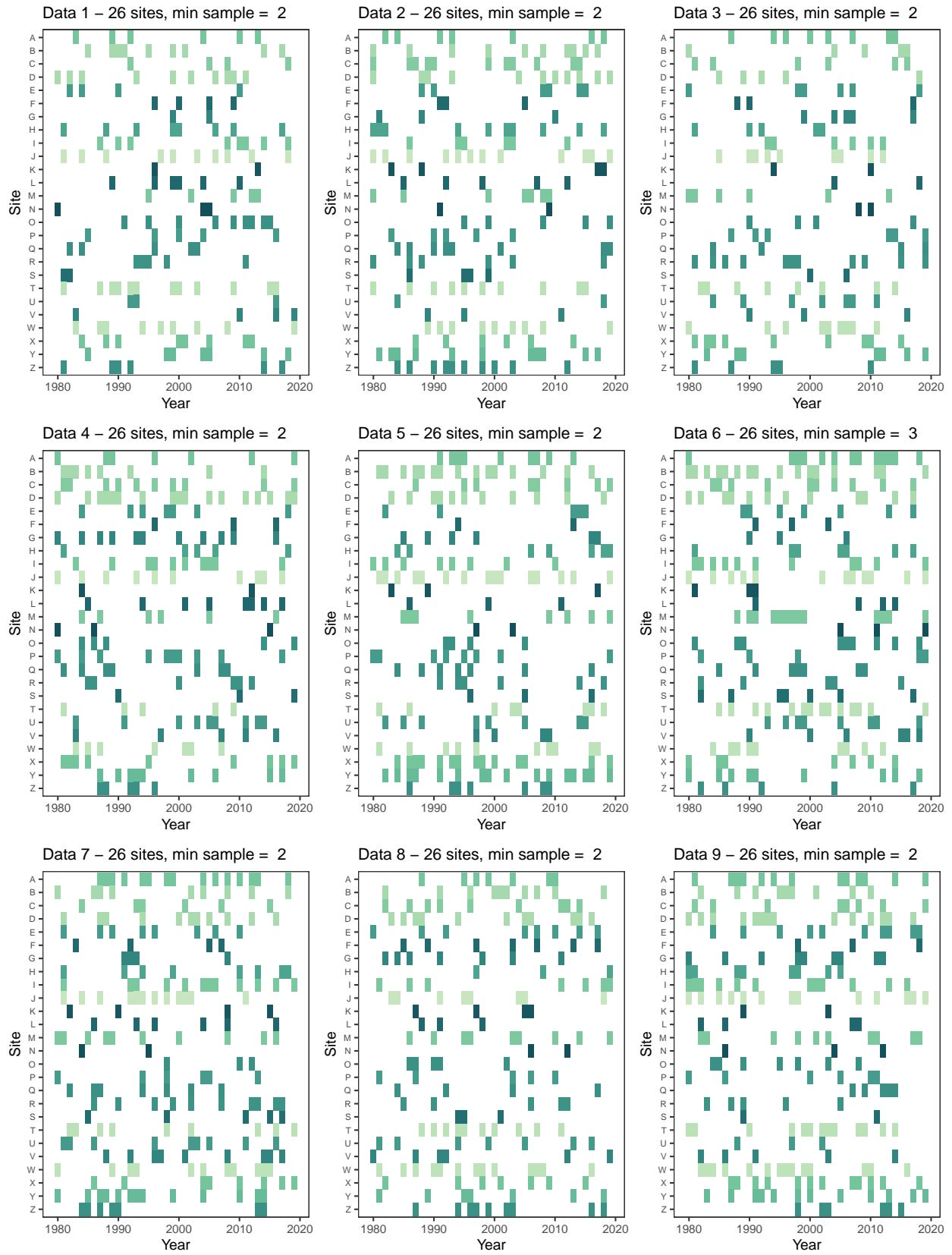
```
}
```

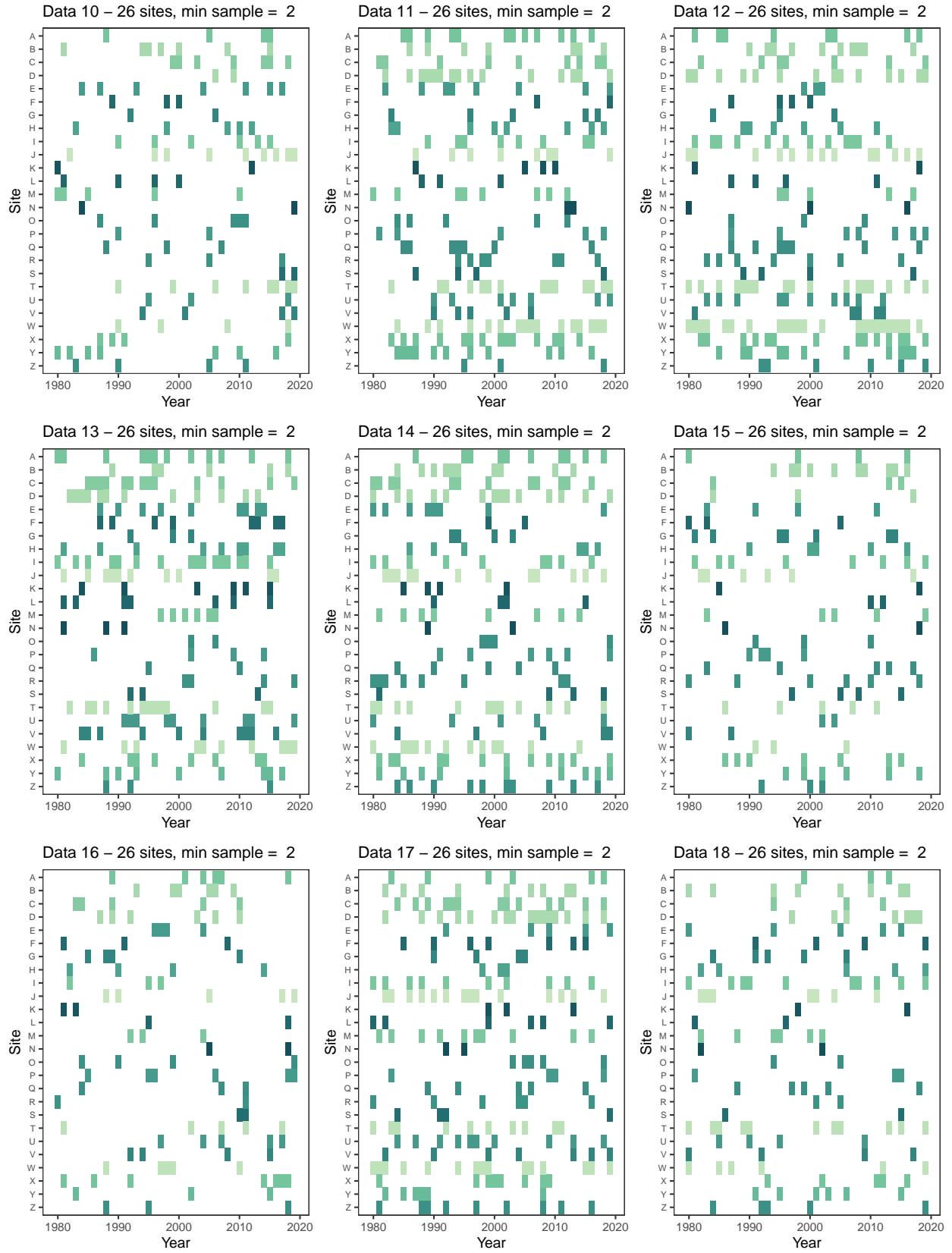
```
## Warning: The '<scale>' argument of 'guides()' cannot be 'FALSE'. Use "none" instead as
## of ggplot2 3.3.4.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```

```
#-----#
# end of loop
#-----#
```

Plot the sparse data time-series ('observed' data)

```
# Arrange and save the plots on the same figure
grid.arrange(grobs=plots_list[1:9], ncol=3); grid.arrange(grobs=plots_list[10:18], ncol=3)
```





```

# Build data frame from simulations of count
sparsedat = bind_rows(list_sparse)

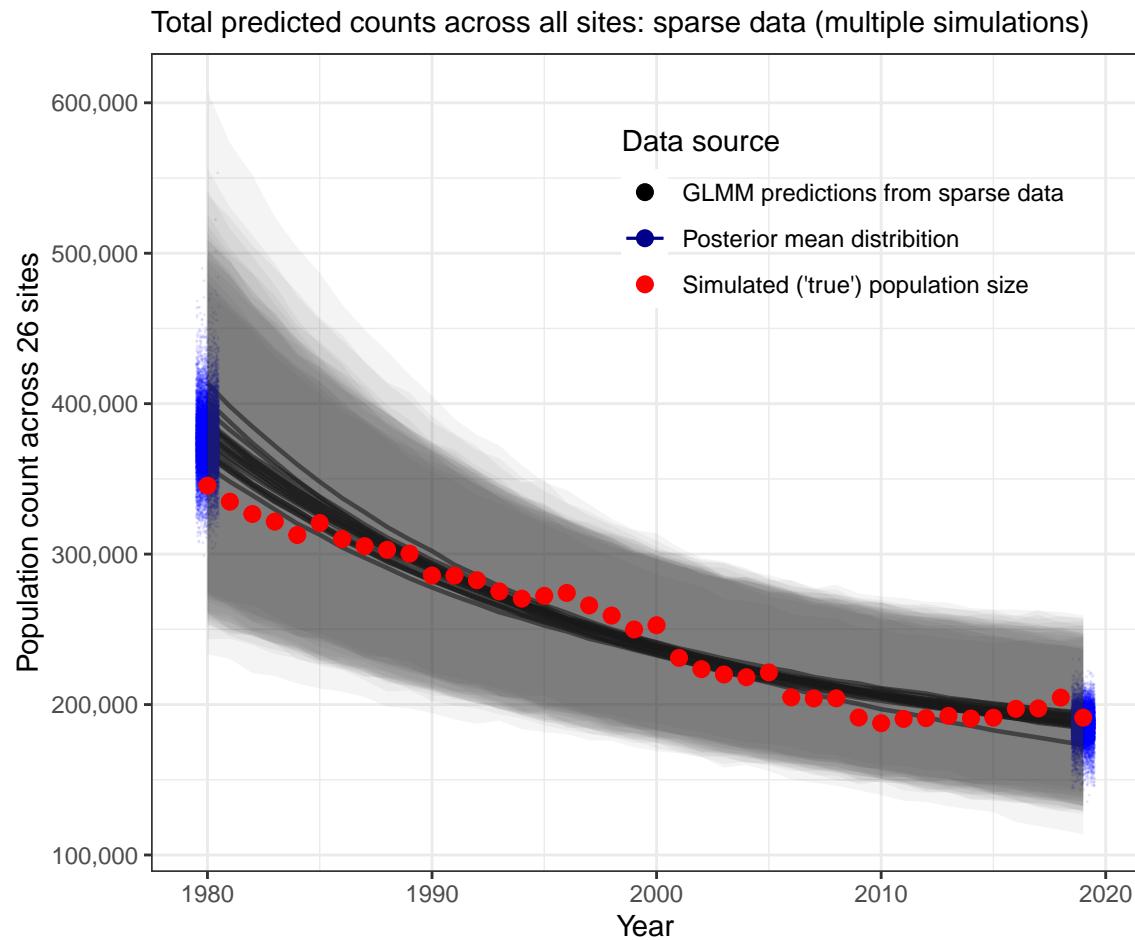
# Create a dummy data point for the legend
legend_data <- data.frame(x = 2000, y = mean(sparsedat$total_pred),
                           color = "GLMM predictions from sparse data")
legend_data2 <- data.frame(x = 2000, y = mean(sparsedat$total_pred),
                           color = "Posterior mean distribution")

# Build data frame from simulations of count
posterior_preds = bind_rows(posterior_list)

# plot
pop_predict.p = ggplot(data = sparsedat) +
  # GLMM predictions (best fit)
  geom_line(aes(x = year, y = total_pred, group = as.factor(iteration)),
            lty = 1, linewidth = 0.8, alpha = 0.6) +
  # posteriors
  geom_jitter(data = posterior_preds, aes(x = as.numeric(year1), y = pred_first),
              color = "blue", size = 0.5, height = 0, width = 0.5,
              alpha = 0.1, stroke = NA) +
  geom_jitter(data = posterior_preds, aes(x = as.numeric(year2), y = pred_last),
              color = "blue", size = 0.5, height = 0, width = 0.5,
              alpha = 0.1, stroke = NA) +
  # GLMM predictions (uncertainty)
  geom_ribbon(aes(x = year, ymin=min_pred, ymax=max_pred, group = as.factor(iteration)),
              alpha=0.05, linetype = 0) +
  labs(x = "Year", y = "Population count across 26 sites") +
  theme_bw() +
  scale_y_continuous(label = comma, breaks = seq(100000, 800000, by = 100000)) +
  scale_x_continuous(breaks = seq(1980, 2020, by = 10),
                     labels = seq(1980, 2020, by = 10)) +
  #labs(subtitle = "Total predicted counts across all sites") +
  guides(color=guide_legend(title="Data source")) +
  theme(legend.position = c(0.7, 0.8)) +
  # Dummy line for legend (does not plot on figure)
  geom_line(data = legend_data, aes(x = x, y = y, color = color),
            linetype = "solid") +
  geom_line(data = legend_data2, aes(x = x, y = y, color = color),
            linetype = "solid") +
  # "Observed" (simulated) true population size
  geom_point(data = population, aes(x = year, y = totalN,
                                    color = "Simulated ('true') population size"), size = 2.5) +
  guides(color = guide_legend(title = "Data source",
                               override.aes = list(
                                 linetype = c("blank", "solid", "blank")))) +
  theme(legend.background = element_rect(fill = "transparent")) +
  scale_color_manual(name = "Data source",
                     values = c("Simulated ('true') population size" = "red",
                               "GLMM predictions from sparse data" = "black",
                               "Posterior mean distribution" = "darkblue")) +
  labs(subtitle = "Total predicted counts across all sites: sparse data (multiple simulations)")

```

```
pop_predict.p
```



```
# "true" population size in 1980
pT = population %>%
  dplyr::filter(year == 1980)
pT
```

```
## # A tibble: 1 x 2
##   year totalN
##   <int>  <dbl>
## 1  1980 345529
```

```
# mean of GLMM predictions
pP = sparsedat %>%
  dplyr::filter(year == 1980) %>%
  summarize(meanpred = mean(total_pred))
pP
```

```
## # A tibble: 1 x 1
##   meanpred
##   <dbl>
## 1 380359.
```

```
# Proportion population size overestimate predicted in 1980  
pP / pT[2]
```

```
## meanpred  
## 1 1.100802
```