Wild Reindeer Nature Index: Reference and Indicator Values

Bart Peeters

2025-04-14

Below is the complete analysis pipeline for calculating reference (K) and indicator values for wild reindeer populations using integrated multi-population model (IMPM) outputs, hunting statistics, Ricker modelling, and reindeer survey data (winter counts).

The original IMPM model outputs in step 1 are not provided in the GitHub repository, but all output files were saved and uploaded to complete steps 2-4. For the next update (2029) of the wild reindeer nature index, the IMPM model outputs should be updated with the latest data, updated IMPM model, and optionally include more populations. If only the indicator values need updating, go directly to step 4. The code includes importing the output data from the previous steps.

```
# Global setup: options and locale encoding
library(here)
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning = FALSE,
eval=FALSE, message=FALSE,
                      root.dir = here())
Sys.setlocale("LC_ALL", "nb-NO.UTF-8")
```

```
## [1] "LC_COLLATE=nb-NO.UTF-8;LC_CTYPE=nb-NO.UTF-8;LC_MONETARY=nb-NO.UTF-
8;LC_NUMERIC=C;LC_TIME=nb-NO.UTF-8"
```

```
# libraries and directories
library(abind)
library(dplyr)
library(tidyr)
library(ggplot2)
library(TMB)
library(sf)
library(openxlsx)

# Define file paths for archived data
input_dir  <- "data_NI/"
output_dir <- "data_NI/"
tmb_dir    <- "TMB_models/"
```

1. Process IMPM Model Outputs

This section loads JAGS-based IMPM outputs for multiple populations, derive summary statistics for total abundance (Xtot), sample consistent posterior simulations, and estimate detectability from minimum counts.

```
# 1.1. Load IMPM JAGS Model Outputs & Harvest Data

# For updated 2024 indicators, models IMPM models were run separately for
Lærdal-Årdal and all other areas.
load("~/PROJECTS/IMPM villrein/Reindeer-
IPM/ModelOutput/model_hypergeom_noDD_multiAreas_2025_500.Rdata")
MP1 <- MP
load("~/PROJECTS/IMPM villrein/Reindeer-
IPM/ModelOutput/model_LardalArdal_2025_200.Rdata")
MP2 <- MP

# Define years and areas from the MP data objects
YEARS   <- 1991:2024
AREAS1  <- rownames(MP1$data$Harv_prop)  # most areas (later: exclude
"Snohetta" if needed)
AREAS2  <- rownames(MP2$data$Harv_prop)
AREAS   <- c(AREAS1, AREAS2)

# Load total harvest data and reshape (both scripts used the same file)
Htot <- read.csv(file = here(input_dir, "reindeer_totalHarvest_data_1991-
2024.csv"),
                 header = TRUE)
Htot <- Htot %>%
  pivot_wider(names_from = area, values_from = htot, names_sort = TRUE) %>%
  select(-year) %>%
  as.data.frame()

# 1.2. Compute Posterior Summaries for Xtot

# Combine posterior means, standard deviations and quantiles from both model
outputs.
Xtot_mean <- cbind(MP1$mean$Xtot, MP2$mean$Xtot)[-1,]
Xtot_sd   <- cbind(MP1$sd$Xtot,   MP2$sd$Xtot)[-1,]
Xtot_q1   <- cbind(MP1$q2.5$Xtot,  MP2$q2.5$Xtot)[-1,]
Xtot_q2   <- cbind(MP1$q97.5$Xtot, MP2$q97.5$Xtot)[-1,]
Xtot_kv25 <- cbind(MP1$q25$Xtot,   MP2$q25$Xtot)[-1,]
Xtot_kv75 <- cbind(MP1$q75$Xtot,   MP2$q75$Xtot)[-1,]

# Set column and row names based on AREAS and YEARS
colnames(Xtot_mean) <- colnames(Xtot_sd) <- colnames(Xtot_q1) <-
colnames(Xtot_q2) <-
  colnames(Xtot_kv25) <- colnames(Xtot_kv75) <- AREAS
rownames(Xtot_mean) <- rownames(Xtot_sd) <- rownames(Xtot_q1) <-
```

```r
rownames(Xtot_q2) <-
  rownames(Xtot_kv25) <- rownames(Xtot_kv75) <- as.character(YEARS)

# Combine into long-form data frame
n_years <- nrow(Xtot_mean)
n_areas <- ncol(Xtot_mean)
df <- data.frame(
  row   = rep(1:n_years, times = n_areas),
  col   = rep(1:n_areas, each = n_years),
  year  = rep(rownames(Xtot_mean), times = n_areas),
  area  = rep(colnames(Xtot_mean), each = n_years),
  mean  = as.vector(Xtot_mean),
  sd    = as.vector(Xtot_sd),
  q1    = as.vector(Xtot_q1),
  q2    = as.vector(Xtot_q2),
  kv25  = as.vector(Xtot_kv25),
  kv75  = as.vector(Xtot_kv75),
  stringsAsFactors = FALSE
)

# Exclude problematic combinations
# - LardalArdal: years 1991–1994 (reintroduced in 1995)
# - SnohettaE/W: years 1991–1992 (poor estimates)
# - Solnkletten: years 1991–2010 (insufficient data)
area_idx <- rbind(
  cbind(which(YEARS %in% 1991:1994), rep(which(AREAS == "LardalArdal"),
length(1991:1994))),
  cbind(which(YEARS %in% 1991:1992), rep(which(grepl("Snohetta", AREAS)),
length(1991:1992))),
  cbind(which(YEARS %in% 1991:2010), rep(which(AREAS == "Solnkletten"),
length(1991:2010)))
)
for (i in 1:nrow(area_idx)) {
  idx <- df$row == area_idx[i,1] & df$col == area_idx[i,2]
  df[idx, c("mean","sd","q1","q2","kv25","kv75")] <- NA
}

# Order by area
df$area <- factor(df$area, levels = sort(unique(df$area)))
df <- df[order(df$area), ]
rownames(df) <- NULL

# Save the posterior summary table:
#write.table(df, file = here(output_dir, "Xtot_output_1991-2024.txt"),
#            sep = "\t", row.names = FALSE, quote = FALSE)

# 1.3. Extract Posterior Samples and Compute Pre-harvest Abundance

# Function to sample simulation runs for Xtot using a criterion:
get_Xtotsim_samples <- function(MP, nSamples = 1000, CI_threshold = 1) {
```

```r
  simX      <- MP$sims.list$Xtot
  lower     <- MP$q2.5$Xtot
  upper     <- MP$q97.5$Xtot
  lower_arr <- aperm(array(lower,
dim=c(dim(simX)[2],dim(simX)[3],dim(simX)[1])), perm=c(3,1,2))
  upper_arr <- aperm(array(upper,
dim=c(dim(simX)[2],dim(simX)[3],dim(simX)[1])), perm=c(3,1,2))
  inside    <- (simX >= lower_arr) & (simX <= upper_arr)
  prop_in   <- apply(inside, c(1,3), mean)
  good      <- prop_in >= CI_threshold
  nPop      <- dim(simX)[3]
  nYear     <- dim(simX)[2]
  sampled   <- array(NA, dim=c(nSamples,nYear,nPop))
  for (pop in 1:nPop) {
    samp_idx <- sample(which(good[,pop]), nSamples)
    sampled[,,pop] <- simX[samp_idx,,pop]
  }
  return(sampled)
}
# Get simulation subsets for both MP objects
set.seed(10)
subset1 <- get_Xtotsim_samples(MP1)
set.seed(10)
subset2 <- get_Xtotsim_samples(MP2)
# Combine along population dimension:
Xtot_sims <- abind(subset1, subset2, along=3)
dimnames(Xtot_sims) <- list(NULL, c(min(YEARS)-1, YEARS), c(AREAS1, AREAS2))
# Remove year 1990 (first column)
Xtot_sims <- Xtot_sims[,-1,]
# Set simulation cells to NA for the excluded year-area combinations defined
earlier:
for (i in 1:nrow(area_idx)) {
  Xtot_sims[,area_idx[i,1],area_idx[i,2]] <- NA
}
# Order populations alphabetically:
Xtot_sims <- Xtot_sims[,,order(AREAS)]

# Save simulation data:
# save(Xtot_sims, file = here(output_dir, "impm_xtot_sims.Rdata"))

# 1.4. Calculate Minimum Count Detectability Estimates

# only use mean and SD detectability from years with minimum count data

# Process detectability probability matrices from both MP outputs.
na1 <- which(is.na(MP1$data$minCount), arr.ind=TRUE)
na2 <- which(is.na(MP2$data$minCount), arr.ind=TRUE)

p1 <- MP1$sims.list$p_minCount
for(i in seq_len(nrow(na1))) p1[,na1[i,1],na1[i,2]] <- NA
```

```r
p2 <- MP2$sims.list$p_minCount
for(i in seq_len(nrow(na2))) p2[,na2[i,1],na2[i,2]] <- NA

# Calculate mean and SD of logit-transformed detectability:
logit <- function(p) log(p/(1-p))
mn1 <- apply(logit(p1), 3, mean, na.rm=TRUE)
sd1 <- apply(logit(p1), 3, sd,   na.rm=TRUE)
mn2 <- apply(logit(p2), 3, mean, na.rm=TRUE)
sd2 <- apply(logit(p2), 3, sd,   na.rm=TRUE)

# Create a data frame with uncertainty per area:
P_minC_uncertainty <- data.frame(Area=AREAS, meanP_minC=c(mn1,mn2),
sdP_minC=c(sd1,sd2)) %>%
  arrange(Area)

# Overall detectability estimates using p_minCount means:
p_mean <- cbind(MP1$mean$p_minCount, MP2$mean$p_minCount)
p_sd   <- cbind(MP1$sd$p_minCount, MP2$sd$p_minCount)
colnames(p_mean) <- colnames(p_sd) <- AREAS
for (i in 1:nrow(area_idx)) {
  p_mean[area_idx[i, 1], area_idx[i, 2]] <- NA
  p_sd[area_idx[i, 1], area_idx[i, 2]] <- NA
}

# Remove year 1990 and order columns alphabetically; also exclude "Snohetta"
p_mean <- p_mean[-1, ]
p_sd   <- p_sd[-1, ]
p_mean <- p_mean[, order(AREAS)]
p_sd   <- p_sd[, order(AREAS)]
p_mean <- p_mean[, colnames(p_mean) != "Snohetta"]
p_sd   <- p_sd[, colnames(p_sd) != "Snohetta"]
# Compute overall average detectability parameters:
p_mean_overall <- colMeans(p_mean, na.rm = TRUE)
p_sd_overall   <- colMeans(p_sd, na.rm = TRUE)
PMEAN <- mean(p_mean_overall)
PSD   <- mean(p_sd_overall)

# Save detectability results:
# save(PMEAN, PSD, file = here(output_dir, "impm_detectability.Rdata"))
```

2. Estimate Carrying Capacities (K)

Using simulated abundance (Xtot) and harvest data from the IMPM output, fit a stochastic Ricker model via TMB to estimate carrying capacity K for each reindeer population in the IMPM. Then fit a linear regression model to estimate the relationship between K and the total reindeer area to predict K for the other wild reindeer areas.

```
### 2.1. Define and Compile the TMB Ricker Model
ricker_code <- '
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() () {
  // DATA
  DATA_VECTOR(N);        // Observed pre-harvest population (time t)
  DATA_VECTOR(X);        // Previous year post-harvest population (time t-1)
  DATA_SCALAR(penalty);  // Small penalty standard deviation

  // RANDOM EFFECTS
  PARAMETER_VECTOR(z);   // Random effects for each time step

  // FIXED PARAMETERS:
  // par[0] = r, par[1] = K, par[2] = log(c)
  PARAMETER_VECTOR(par);
  Type r = par[0];
  Type K = par[1];
  Type c = exp(par[2]);

  // Initialize negative log-likelihood:
  Type nll = 0.0;

  // Prior for random effects: z ~ N(0,1)
  nll -= dnorm(z, Type(0), Type(1), true).sum();

  // Predicted N using the Ricker formulation with additive noise on the
growth term:
  vector<Type> pred_N(N.size());
  for(int i = 0; i < N.size(); i++){
    pred_N(i) = X(i) * exp(r * (1.0 - X(i) / K) + c * z(i));
  }

  // Likelihood term: force pred_N to be near the observed N with a small
penalty SD.
  nll -= dnorm(pred_N, N, penalty, true).sum();

  return nll;
}
'
# Write the TMB model code to file, compile, and load the DLL
write(ricker_code, file = here(tmb_dir, "tmb_Ricker_addnoiseGrowth.cpp"))
compile(here(tmb_dir, "tmb_Ricker_addnoiseGrowth.cpp"))
dyn.load(dynlib(here(tmb_dir, "tmb_Ricker_addnoiseGrowth")))

### 2.2 Prepare Data for TMB Analysis
# Exclude "Snohetta" if desired (to match harvest data and simulations)
Htot <- Htot[, !colnames(Htot) %in% c("Snohetta")]
Xtot_sims <- Xtot_sims[, , dimnames(Xtot_sims)[[3]] %in% colnames(Htot)]
```

```r
AREAS <- dimnames(Xtot_sims)[[3]]
YEARS <- dimnames(Xtot_sims)[[2]]
# For "Nordfjella", use only data prior to 2017:
Xtot_sims[, which(YEARS >= 2017), which(AREAS == "Nordfjella")] <- NA

# Compute the average posterior estimates from simulation runs and derive
pre-harvest abundance.
Xtot <- apply(Xtot_sims, c(2, 3), mean)
Ntot <- Xtot + Htot  # Ntot = post-harvest + total harvest

# 2.3 Fit the model for each area
# Helper function to Fit the Ricker Model with TMB
fit_ricker <- function(N, X, penalty = 1e-3, init_r = 0.3,
                       init_K = 1.2 * max(N, na.rm = TRUE), init_logC = -1) {
  data_list <- list(N = N, X = X, penalty = penalty)
  params_list <- list(
    z   = rep(0, length(N)),        # random effects initialization
    par = c(init_r, init_K, init_logC)  # parameters: r, K, log(c)
  )
  obj <- MakeADFun(data = data_list, parameters = params_list,
                   random = "z", DLL = "tmb_Ricker_addnoiseGrowth", silent =
TRUE,
                   newtonsteps = 2)
  obj$hessian <- FALSE
  obj$control <- list(maxit = 100000)
  opt <- optim(par = obj$par, fn = obj$fn, gr = obj$gr,
               method = "BFGS", control = list(maxit = 10000, reltol = 1e-
10))
  return(list(opt = opt, obj = obj))
}

# fit the model for each area
list_models <- list()
for(pop in AREAS){
  idx <- which(AREAS == pop)
  N <- Ntot[-1, idx]      # Observed pre-harvest population (from time t)
  X <- Xtot[-nrow(Xtot), idx]  # Previous year post-harvest population
  df_mod <- data.frame(N = N, X = X)
  df_mod <- df_mod[complete.cases(df_mod), ]
  list_models[[pop]] <- fit_ricker(df_mod$N, df_mod$X)
}
# Optionally inspect model diagnostics:
#lapply(list_models, function(x) sdreport(x$obj))

# 2.4 Derive K statistics
### Loop Over Posterior Simulations to Derive K Estimates
nSims <- dim(Xtot_sims)[1]
K_sims_Ricker <- array(NA, dim = c(nSims, length(AREAS)))
dimnames(K_sims_Ricker) <- list(NULL, AREAS)
for(pop in AREAS){
```

```r
  idx <- which(AREAS == pop)
  for(i in 1:nSims){
    X_i <- Xtot_sims[i, , idx]
    # Pre-harvest abundance: simulation + harvest
    N_i <- X_i + Htot[, idx]
    mod_df <- data.frame(N = N_i[-1], X = X_i[-length(X_i)])
    mod_df <- mod_df[complete.cases(mod_df), ]
    if(nrow(mod_df) > 0){
      model_fit <- fit_ricker(mod_df$N, mod_df$X)
      rep_fit <- sdreport(model_fit$obj)
      # The second parameter (K) is extracted from the report summary:
      K_sims_Ricker[i, idx] <- unname(summary(rep_fit)[2, 1])
    }
  }
}
# Summarize the K estimates per area:
K_results <- data.frame(
  omrade   = AREAS,
  K_mean   = apply(K_sims_Ricker, 2, mean, na.rm = TRUE),
  K_median = apply(K_sims_Ricker, 2, median, na.rm = TRUE),
  K_sd     = apply(K_sims_Ricker, 2, sd, na.rm = TRUE)
)
# Write K results summary to file:
#write.table(K_results, file = here(output_dir, "K_results_Ricker.txt"),
#            row.names = FALSE, quote = FALSE, sep = "\t")
```

3. Compute Reference Values (K) For Each Reindeer Area

This section fits a linear regression model of K and the total reindeer area, to predict K for other wild reindeer areas. The model is fitted using the K estimates derived from the TMB Ricker model in step 2 using the population estimates from step 1 and hunting statistics.

```r
### 3.1. Load Reindeer Area Polygons and Adjust Area Measures
omrader <- st_read(here(input_dir, "reindeer_areas_inc_subdivision.gpkg"))
%>%
  mutate(Areal = ifelse(is.na(Areal), Areal_ny_km2, Areal)) |>
  filter(!Villreinom  %in% c("Nordfjella Sone 1", "Nordfjella Sone 2",
"Snøhetta","Rondane","Tolga østfjell"))

# Select and rename columns; adjust area names to ASCII
omrader <- as.data.frame(villreinomrader) %>%
  select(omrade = Villreinom, areal = Areal) %>%
  mutate(omrade = case_when(
    omrade == "Snøhetta øst" ~ "SnohettaE",
    omrade == "Snøhetta vest" ~ "SnohettaW",
    omrade == "Rondane nord"  ~ "RondaneN",
    omrade == "Rondane sor"    ~ "RondaneS",
    TRUE ~ omrade
  ))
```

```r
omrader$omrade <- gsub("æ", "ae", omrader$omrade)
omrader$omrade <- gsub("ø", "o",  omrader$omrade)
omrader$omrade <- gsub("å", "a",  omrader$omrade)
omrader$omrade <- gsub("Æ", "Ae", omrader$omrade)
omrader$omrade <- gsub("Ø", "O",  omrader$omrade)
omrader$omrade <- gsub("Å", "A",  omrader$omrade)

### 3.2. Join K Estimates with Area Data and Project K for Other Areas
# load data (estimates from section K)
K_results <- read.table(here(output_dir, "K_results_Ricker.txt"),
                        header = TRUE, sep = "\t")
# Adjust the name for consistency (e.g., "LardalArdal" becomes "Laerdal-
Ardal")
K_results <- K_results %>%
  mutate(omrade = ifelse(omrade == "LardalArdal", "Laerdal-Ardal", omrade))
# Join with area data:
K_df <- left_join(omrader, K_results, by = "omrade") %>%
  rename(K = K_mean) %>%
  arrange(desc(areal))

# Fit a linear model: log(K) ~ log(areal)
logmod <- lm(log(K) ~ log(areal), data = K_df)
# (Optional) Examine model diagnostics:
# summary(logmod)
# par(mfrow = c(2, 2)); plot(logmod); par(mfrow = c(1,1))

# For simulation-based uncertainty, run predictions over each of 1000 K
simulations:
load(here(output_dir,"K_sims_Ricker.RData"))
nSims = dim(K_sims_Ricker)[1]
# (Assuming K_sims_Ricker has the same area names; adjust name if necessary)
colnames(K_sims_Ricker)[colnames(K_sims_Ricker) == "LardalArdal"] <-
"Laerdal-Ardal"
pop_names <- colnames(K_sims_Ricker)
areas <- K_df %>% filter(omrade %in% pop_names) %>% arrange(omrade)
areal <- areas %>% pull(areal)
pred_logK <- matrix(NA, nrow = nSims, ncol = nrow(K_df))
K_df <- K_df %>% arrange(omrade)
for (sim_i in 1:nSims) {
  K_i <- K_sims_Ricker[sim_i, ]
  df_i <- data.frame(K = K_i, areal = areal)
  mod_i <- lm(log(K) ~ log(areal), data = df_i)
  pred_logK[sim_i, ] <- predict(mod_i, newdata = K_df)
}
pred_K <- exp(pred_logK)
K_df$pred_K_mean <- apply(pred_K, 2, mean)
K_df$pred_K_sd   <- apply(pred_K, 2, sd)
K_df$pred_K_q1   <- exp(apply(pred_logK, 2, quantile, 0.025))
K_df$pred_K_q2   <- exp(apply(pred_logK, 2, quantile, 0.975))
K_df$pred_K_Q25  <- exp(apply(pred_logK, 2, quantile, 0.25))
```

```r
K_df$pred_K_Q75  <- exp(apply(pred_logK, 2, quantile, 0.75))

# Save reference predictions
#write.table(K_df, file = here(output_dir, "K_predictions.txt"),
#            sep = "\t", row.names = FALSE, quote = FALSE)
```

4. Calculate Indicator Values

Minimum count estimates are used to calculate the annual indicator values for each population. For NI years (1990, 2000, 2010, 2014, 2019, 2024) with missing count data, winter abundance is estimates as the average counts from 1, 2 or 3 years before and after the NI year.

```r
### 4.1. Load or Use Population Estimates and Detectability

# load long-torm Xtot data (from step 1)
df <- read.table(here(output_dir, "Xtot_output_1991-2024.txt"),
             header = TRUE, sep = "\t")
# Load K values from TMB (from step 2)
K_results <- read.table(here(output_dir, "K_results_Ricker.txt"),
                        header = TRUE, sep = "\t")
# load K predictions from log linear model (from step 3)
K_df <- read.table(here(output_dir, "K_predictions.txt"),
                   header = TRUE, sep = "\t")

# Load detectability probability mean and SD from step 1
load(here(output_dir, "impm_detectability.Rdata"))
PMEAN
PSD

# Read Minimum Count Data and Adjust Names
min_counts <- openxlsx::read.xlsx(here(input_dir, "Stammeregnskap for
Naturindeks25_BBH_20250316.xlsx")) %>%
  rename(omrade = Pop, year = Year) %>%
  select(-Source) %>%
  filter(!omrade %in% c("Lærdal-Årdal", "Tolga ostfjell"))  # Exclude if
necessary
# Replace Norwegian letters with ASCII equivalents
min_counts$omrade <- min_counts$omrade %>%
  gsub("æ", "ae", .) %>%
  gsub("ø", "o", .) %>%
  gsub("å", "a", .) %>%
  gsub("Æ", "Ae", .) %>%
  gsub("Ø", "O", .) %>%
  gsub("Å", "A", .)

### 4.2 Calculate Indicator Values
```

```r
# Derive Population Estimates as Minimum Counts adjusted by Detectability
min_counts <- min_counts %>%
  mutate(N_est = Min_N / PMEAN,
         N_sd = Min_N * PSD / PMEAN^2)

### Combine Model-based and Minimum Count Estimates
# Rename Xtot estimates to match minimum count data and bind the two sources
indicator <- df %>%
  rename(N_est = mean, N_sd = sd, omrade = area) %>%
  select(-row, -col) %>%
  bind_rows(min_counts) %>%
  mutate(omrade = ifelse(omrade == "LardalArdal", "Laerdal-Ardal", omrade),
         N_est = ifelse(is.na(N_est) & omrade == "Laerdal-Ardal", 0, N_est),
         N_sd  = ifelse(N_est == 0, 0, N_sd),
         kv25  = N_est - N_sd * 0.674,
         kv75  = N_est + N_sd * 0.674)

### Impute Missing Indicator Years
# Define target (NI) years to impute indicators:
target_years <- c(1990, 2000, 2010, 2014, 2019, 2024)
# Imputation function: For missing target years, take the average of the
closest available years.
impute_abundance <- function(ind_df) {
  ind_df <- ind_df %>%
    mutate(N = N_est, SD = N_sd,
           Q25 = kv25, Q75 = kv75) %>%
    group_by(omrade) %>%
    tidyr::complete(year = tidyr::full_seq(c(1990, max(year, na.rm = TRUE)),
1)) %>%
    arrange(omrade, year)

  for (yr in target_years) {
    ind_df <- ind_df %>%
      group_by(omrade) %>%
      mutate(
        N = ifelse(year == yr & is.na(N),
                   coalesce(
                     rowMeans(cbind(lag(N, 1), lead(N, 1)), na.rm = TRUE),
                     rowMeans(cbind(lag(N, 2), lead(N, 2)), na.rm = TRUE),
                     lag(N, 3),
                     lead(N, 3)
                   ),
                   N),
        SD = ifelse(year == yr & is.na(SD),
                    coalesce(
                      rowMeans(cbind(lag(SD, 1), lead(SD, 1)), na.rm = TRUE),
                      rowMeans(cbind(lag(SD, 2), lead(SD, 2)), na.rm = TRUE),
                      lag(SD, 3),
                      lead(SD, 3)
                    ),
```

```
                    SD),
        Q25 = ifelse(year == yr & is.na(Q25),
                     coalesce(
                       rowMeans(cbind(lag(Q25, 1), lead(Q25, 1)), na.rm =
TRUE),
                       rowMeans(cbind(lag(Q25, 2), lead(Q25, 2)), na.rm =
TRUE),
                       lag(Q25, 3),
                       lead(Q25, 3)
                     ),
                     Q25),
        Q75 = ifelse(year == yr & is.na(Q75),
                     coalesce(
                       rowMeans(cbind(lag(Q75, 1), lead(Q75, 1)), na.rm =
TRUE),
                       rowMeans(cbind(lag(Q75, 2), lead(Q75, 2)), na.rm =
TRUE),
                       lag(Q75, 3),
                       lead(Q75, 3)
                     ),
                     Q75)
      ) %>%
      ungroup()
  }
  return(ind_df)
}

df_impute <- impute_abundance(indicator)

# Merge with reference values (K), calculate the ratio indicator:
referanse <- K_df %>%
  mutate(verdi = ifelse(is.na(K), pred_K_mean, K),
         nedre_Kvartil = verdi - 0.674 * ifelse(is.na(K), pred_K_sd,
K_results$K_sd),
         ovre_Kvartil  = verdi + 0.674 * ifelse(is.na(K), pred_K_sd,
K_results$K_sd),
         yearName = "Referanseverdi") %>%
  select(omrade, areal, verdi, nedre_Kvartil, ovre_Kvartil, yearName)

# Prepare indicator time series for target years only:
df_NI <- df_impute %>%
  mutate(yearName = as.character(year)) %>%
  select(omrade, yearName, verdi = N_est,
         nedre_Kvartil = Q25, ovre_Kvartil = Q75) %>%
  bind_rows(referanse) %>%
  arrange(omrade, desc(yearName))

# Plot the indicator time series:
gin <- df_impute %>%
```

```
  left_join(referanse, by = "omrade") %>%
 # filter(year %in% target_years) %>%
  mutate(u1 = Q25 / verdi, u2 = Q75 / verdi) %>%
  ggplot(aes(x = year, y = N / verdi, col = omrade)) +
  geom_line() +
  geom_point() +
  geom_errorbar(aes(ymin = u1, ymax = u2)) +
  theme_bw() + theme(legend.position = "none") +
  facet_wrap(~ omrade, scales = "free_y") +
  ylim(0, 1.6) +
  geom_hline(yintercept = c(0, 1), col = "red")

print(gin)



# Save final indicator summary table:
#write.table(df_NI, file = here(output_dir,
"newIndicator_Villrein_2024.txt"),
#            sep = "\t", row.names = FALSE)
```

5. Upload New Indicator Values to the Nature Index

Script to upload new values to NI. The Helper functions from the NI team are provided and minorly adjusted from the original script. The Reference values and indicator data are estimated as number of reindeer, but included total area in the output so that density can be calculated for uploading in NI.

```
### 5.1 NIcalc package and helper functions
if(!("NIcalc" %in% installed.packages())){
  devtools::install_github("NINAnor/NIcalc", build_vignettes = F)
}
library(NIcalc)
library(magrittr)
library(dplyr)

# Helper functions: ------------------------------------------------------
--
#' Download indicator data from the Nature Index database
#'
#' This function retrieves the currently stored values for one or several
#' indicators from the Nature Index (NI) database using the function
#' `getIndicatorValues` from the `NICalc` package.
#'
#' Executing this function requires accessing the NI database through a token
#' generated by `getToken`, and users are prompted to enter their NI database
#' log-in credentials for that purpose. NI database credentials consist of a
#' registered email address and a password.
```

```r
#'
#' @param indicators A vector of character strings representing the Norwegian
#' names (without special characters) of the relevant indicators. The
#' spelling of `indicators` is consistent with the spelling used in the NI
#' database (field "Name").
#' @param save Logical, default = `FALSE`. If `TRUE`, the downloaded
indicator
#' data is saved in the workspace in a file named `oldIndicatorData.rds`.
#' @param save_path Character string indicating the directory into which to
save
#' the old indicator data. Defaults to the current working directory.
#'
#' @return A list of objects (one per entry in `indicators`) containing 1) a
data frame of
#' the values of the values of the indicator and 2) a list of distribution
#' objects quantifying the uncertainty of each indicator value estimate.
#' @export
#'
#' @examples
#'
downloadData_NIdb <- function(indicators, save = FALSE, save_path = getwd()){

  ## Provide user credentials for NI database and request token
  UserName_NIdb <- rstudioapi::askForPassword("NI database username") # =
NINA email address
  Password_NIdb <- rstudioapi::askForPassword("NI database password")

  ## Get token
  NIcalc::getToken(username = UserName_NIdb,
                   password = Password_NIdb,
                   url = "https://www8.nina.no/NaturindeksNiCalc")

  ## Specify indicators for which to download data
  myIndicators <- NIcalc::getIndicators() %>%
    dplyr::right_join(data.frame(name = indicators), by = 'name')

  ## Check if user has access to indicator
  if(any(is.na(myIndicators$id))){
    stop("You do not have access to the requested indicator(s).
         To request access, contact NINA.")
  }

  ## Retrieve old indicator data from NI database
  oldIndicatorData <- list()

  for(i in 1:length(indicators)){
    oldIndicatorData[[i]] <- NIcalc::getIndicatorValues(indicatorID =
myIndicators$id[i])
  }
```

```r
  names(oldIndicatorData) <- myIndicators$name

  ## Save indicator data (optional)
  if(save){
    saveRDS(oldIndicatorData, file = here(save_path,
"/oldIndicatorData.rds"))
  }

  ## Return indicator data
  return(oldIndicatorData)
}

#' Write updated indicator data to the Nature Index (NI) database
#'
#' The function uploads the newly assembled indicator data to the NI
database.
#' Provided the user has writing access, this will overwrite the data that is
#' currently in the database and should ONLY be done when the database needs
to
#' be updated, and after the updated indicator data has been quality-checked.
#'
#' When attempting to write to the NI database, the used will be prompted to
#' confirm the action prior to its execution.
#'
#' Executing this function requires accessing the NI database through a token
#' generated by `getToken` from the `NIcalc` package, and users are prompted
to
#' enter their NI database log-in credentials for that purpose. NI database
#' credentials consist of a registered email address and a password.
#'
#' @param indicatorData a list containing updated indicator data in the same
#' format as data downloaded from the Nature index database.
#'
#' @return
#' @export
#'
#' @examples
uploadData_NIdb <- function(indicatorData){

  ## Provide user credentials for NI database and request token
  myUserName_NIdb <- rstudioapi::askForPassword("NI database username")
  myPassword_NIdb <- rstudioapi::askForPassword("NI database password")

  NIcalc::getToken(username = myUserName_NIdb,
                   password = myPassword_NIdb,
                   url = "https://www8.nina.no/NaturindeksNiCalc"
  )

  # Ask the user for confirmation to write to database
```

```
  command <- askYesNo("Do you want to write to the database?", default =
FALSE)

  # Write to database if confirmed (halt execution otherwise)
  if(!is.na(command) & command){

    message("Uploading new indicator data to NI database:")

    for(i in 1:length(indicatorData)){
      message(names(indicatorData)[i])
      NIcalc::writeIndicatorValues(indicatorData[[i]])
    }
  }else{
    message("Function halted.")
  }
}
```

5.2 Upload new indicator values to the NI database

```
# Load the indicator data - need to log in
oldIndicatorData <- downloadData_NIdb(indicators = "Villrein")
#str(oldIndicatorData$Villrein,1)
#head(oldIndicatorData$Villrein$indicatorValues)

# old area names
oldAreas <- unique(oldIndicatorData[[1]]$indicatorValues$areaName)

# save old data
#saveRDS(oldIndicatorData, file = "output/oldIndicatorData_1990-2019.rds")

# Key points for updating values
# What should be the same for all data points, including the reference value,
is unitOfMeasurement.
# nedre_Kvartil and ovre_Kvartil are the 25% and 75% quartiles of the
uncertainty distribution for the indicator value

# need to calculate tetthet villrein (per km2): for reference value K / areal

# load values
df_NI <- read.table("output/newIndicator_Villrein_2024.txt",header =
T,sep="\t") |>
  filter(!omrade %in% c("Tolga ostfjell","Rondane","Snohetta",
                        "Nordfjella Sone 1","Nordfjella Sone 2"),
         yearName != "2025") |>
  mutate(areaName =
           case_when(
             omrade == "Fordefjella" ~ "Førdefjella",
             omrade == "Knutsho" ~ "Knutshø",
             omrade == "Norefjell-Reinsjofjell" ~ "Norefjell-Reinsjøfjell",
```

```
            omrade == "Vamur-Roan" ~ "Våmur-Roan",
            omrade == "Oksenhalvoya" ~ "Oksenhalvøya",
            omrade == "Laerdal-Ardal" ~ "Lærdal-Årdal",
            omrade == "RondaneN" ~ "Rondane Nord",
            omrade == "RondaneS" ~ "Rondane Sør",
            omrade == "SnohettaE" ~ "Snøhetta Øst",
            omrade == "SnohettaW" ~ "Snøhetta Vest",
            omrade == "Solnkletten" ~ "Sølnkletten",
            omrade == "Vamur-Roan" ~ "Våmur-Roan",
            .default = omrade
          )
  ) |>
  select(-omrade)

# check that names are the same
cbind(sort(oldAreas),sort(unique(df_NI$areaName)),sort(oldAreas)
==sort(unique(df_NI$areaName)))

# change to animals per km2 (density).
df_NI_tetthet <- df_NI |>
  group_by(areaName) %>%
  mutate(areal = if_else(is.na(areal), first(areal[yearName ==
"Referanseverdi"]), areal),
         verdi = verdi / areal,
         nedre_Kvartil = nedre_Kvartil / areal,
         ovre_Kvartil = ovre_Kvartil / areal,
         datatype = ifelse(is.na(verdi), NA, 3) # beregnet fra modeller, NA
if no data available
         ) |>
  select(-areal)

# copy old data
updatedIndicatorData <- oldIndicatorData

# get area metadata
meta_area <- oldIndicatorData[[1]]$indicatorValues[,c("areaName","areaId")]
|> distinct()
# loop through data.frame
for(i in 1:nrow(df_NI_tetthet)){
  row <- df_NI_tetthet[i,]
  updatedIndicatorData[[1]] <-
NIcalc::setIndicatorValues(updatedIndicatorData[[1]],
                                    areaId =
meta_area$areaId[meta_area$areaName == row$areaName],
                                    years = row$yearName,
                                    est = row$verdi,
                                    lower = row$nedre_Kvartil,
                                    upper = row$ovre_Kvartil,
                                    datatype = row$datatype,
```

```
                                    unitOfMeasurement = "Tetthet villrein (per
km2)"
                                    )
}

# Check available data
updatedIndicatorData[[1]]$indicatorValues %>%
   View()

# Compare object classes & dimensions
class(oldIndicatorData) == class(updatedIndicatorData)
length(oldIndicatorData) == length(updatedIndicatorData)
names(oldIndicatorData) == names(updatedIndicatorData)


## Make a data frame containing information on changed fields
diffData <- oldIndicatorData[[1]]$indicatorValues[, 1:6]
colnames(diffData) <- c("indicatorId_meta", "indicatorName_meta",
"areaId_meta", "areaName_meta", "yearId_meta", "yearName_meta")

for(i in 1:ncol(oldIndicatorData[[1]]$indicatorValues)){

  diffData$focal_col <- paste(oldIndicatorData[[1]]$indicatorValues[, i]) ==
paste(updatedIndicatorData[[1]]$indicatorValues[, i])
  diffData <- diffData %>%
    dplyr::mutate(focal_col = ifelse(focal_col, "x", "updated"))
  colnames(diffData)[which(colnames(diffData) == "focal_col")] <-
colnames(oldIndicatorData[[1]]$indicatorValues)[i]
}

## Show overview over all changes
diffData$Updated <- "no"
for(i in 1:nrow(diffData)){

  if("updated" %in% diffData[i, ]){
    diffData$Updated[i] <- "yes"
  }
}

diffData %>%
  dplyr::filter(Updated == "yes")


# UPLOAD - need to be logged in to the NI database
uploadData_NIdb(indicatorData = updatedIndicatorData)

# save
#write.csv(updatedIndicatorData[[1]]$indicatorValues,
```

```
"output/updatedIndicatorData.csv", row.names = F)
#saveRDS(updatedIndicatorData, file = "output/updateddIndicatorData_1990-
2024.rds")
```