Supporting Information. Overcoming data gaps using integrated models to estimate migratory species' dynamics during cryptic periods of the annual cycle

Supplementary S1: Additional modeling details

This supplement provides further description of the spatial random effect in eqn. 1 of the main text. We also included the equations for the log-likelihoods for both presence-only and single-visit count data. Here, we define parameters not within the main text; otherwise, parameter definitions are found within the main text.

Spatial random effect

We specified a spatial random effect in eqn. 1 (main text) using a Gaussian random field to capture heterogeneity in abundance not explained by covariates. The Gaussian random field is described using a multivariate normal distribution, $\omega \sim MVN(0, \tau Q)$ and implemented using a stochastic partial differential equation approximation via a triangulated spatial mesh (which may vary by time period) of $k = 1, \ldots, K_t$ nodes across S_t (Simpson et al. 2016, Krainski et al. 2018). Here, τ is the precision hyperparameter (i.e., inversely related to variance), and Q is a matrix that describes the correlation structure across S_t. We specify correlations between locations in S_t using a Matérn correlation function with scale parameter κ and fixed smoothness parameter v (Simpson et al. 2016, Krainski et al. 2018), which indicates that correlations among $\lambda_t(s)$ values are driven by spatial proximity, with locations near one another having similar intensities (i.e., expected number of individuals).

Log-likelihood

The log-likelihood function for the presence-only data is:

$$\ell_{PO}(\lambda_0, \boldsymbol{\beta}, \delta_t, \omega, p_0, \boldsymbol{\alpha}) =$$
$$\sum_{t=1}^{T}\left(-\int_{S_t}\frac{\lambda_0 \cdot \exp\left(\boldsymbol{\beta}'\boldsymbol{X}_t(s)+\delta_t+\omega(s)+p_0+\boldsymbol{\alpha}'\boldsymbol{W}_t(s)\right)}{\exp\left(p_0+\boldsymbol{\alpha}'\boldsymbol{W}_t(s)\right)+1}ds + \sum_{i=1}^{Y_t}\left(\log(\lambda_0) + \delta_t + \boldsymbol{\beta}'\boldsymbol{X}_t(s_i) + \omega(s_i) + \right.\right.$$
$$\left.\left. p_0 + \boldsymbol{\alpha}'\boldsymbol{W}_t(s_i) - \log\left(\exp\left(p_0 + \boldsymbol{\alpha}'\boldsymbol{W}_t(s_i)\right) + 1\right)\right)\right).$$

Within a single period t, the log-likelihood can be decomposed in two parts. The first is the integral across domain S_t for the thinned process. To approximate the integral over S_t, the thinned process is estimated at each of the k mesh nodes, and a weighted sum is calculated where each node is provided a spatial (i.e., area) weight w_k (Krainski et al. 2018):

$$\int_{S_t}\lambda(s)p(s)ds \approx \sum_{k=1}^{K} w_k\lambda_k p_k.$$

The second part of the log-likelihood is the contribution of all the presence-only observations $i = 1 \ldots, Y_t$ at period t for each of the locations s_i. A projection matrix, A_{ki} is needed to interpolate the random effects ω_i from each node k to each presence-only observation i at location s_i. The covariates, $\boldsymbol{X}_t$ and $\boldsymbol{W}_t$ are extracted for each node k in addition to the presence-only locations s_i.

The log-likelihood function for count data follows a Poisson distribution:

$$\ell_C(\lambda_0, \boldsymbol{\beta}, \delta_t, \omega) = \sum_{t=1}^{T}\left(\sum_{j=1}^{J_t} \log\left(\frac{\exp\left(\log(\lambda_0)+\boldsymbol{\beta}'\boldsymbol{X}_{tj}+\delta_t+\omega_j+\log(D_j)\right)\cdot\left(\log(\lambda_0)+\boldsymbol{\beta}'\boldsymbol{X}_{tj}+\delta_t+\omega_j+\log(D_j)\right)^{C_{tj}}}{C_{tj}!}\right)\right).$$

We note that data gaps in structured sampling for a given time period t, would lead to $J_t = 0$ (i.e., no structured sampling sites) with no data informing the log-likelihood here. The offset for the area for each site, D_j, is standardized to equal one when the area of a site is equal to the mean.

To form the joint-likelihood, we specify the product of the likelihoods of the structured (e.g., count) and unstructured (e.g., presence-only) data:

$$L_{IDM}(\lambda_0, \boldsymbol{\beta}, \delta_t, \omega, p_0, \boldsymbol{\alpha}) = L_C(\lambda_0, \boldsymbol{\beta}, \delta_t, \omega) \cdot L_{PO}(\lambda_0, \boldsymbol{\beta}, \delta_t, \omega, p_0, \boldsymbol{\alpha}).$$

Supplementary S2: Simulation study code and output

This supplement contains additional results of the simulation study as well as the R and C++ code to run the simulation study evaluating the integrated model. Please also refer to archived repositories on either Zenodo (DOI: 10.5281/zenodo.8433370) or GitHub (https://github.com/zipkinlab/Farr_etal_2024_MEE).

Simulation output

Table S2.1. Simulated values, estimated values (mean, 95% C.I.), and percent relative bias (median, lower and upper quartiles) for both the integrated (I) and presence-only (PO) analyses. Estimates were not provided if simulated values were not fixed across simulations.

Parameters	Model	Mean estimate (95% C.I.)	Percent relative bias
$p_0 = -4.59$	I	-4.60 (-5.68, -3.80)	-0.43 (-6.09, 5.43)
	PO	-8.97 (-21.15, -2.90)	5.11 (-14.53, 283.10)
$\alpha = 0.85$	I	0.84 (0.54, 1.21)	-1.18 (-12.94, 10.59)
	PO	1.04 (0.56. 1.89)	12.94 (-5.88, 38.82)
$\lambda_0 = 6.00$	I	6.10 (5.82, 6.37)	1.67 (0.17, 3.00)
	PO	10.14 (4.05, 22.19)	-4.75 (-17.33, 220.38)
$\beta \sim U(0.50,1.50)$	I	Not fixed	-2.99 (-12.66, 8.08)
	PO		-2.94 (-14.50, 9.23)
$\delta = 1.00$	I	1.00 (0.72, 1.27)	0.00 (-6.00, 6.00)
	PO	0.99 (0.68, 1.31)	0.00 (-7.00, 6.00)
$\kappa = 2.36$	I	3.66 (0.95, 10.47)	14.83 (-23.31, 95.44)
	PO	0.71 (0.35, 1.33)	-71.61 (-77.97, -63.14)
$\tau = 0.27$	I	0.17 (0.02, 0.35)	-37.04 (-70.37, -11.11)
	PO	0.26 (0.19, 0.35)	-3.70 (-14.81, 3.70)
$N_1 \approx 3600$	I	Not fixed	1.31 (0.64, 1.92)
	PO		-33.05 (-67.68, 5.27×10^7)
$N_2 \approx 9900$	I	Not fixed	1.22 (-2.88, 5.29)
	PO		-32.83 (-67.63, 5.42×10^7)

R code for the simulation study

```
#-----------#
#-Libraries-#
#-----------#

library(spatstat)
library(RandomFields)
library(INLA)
library(rgeos)
library(TMB)
library(gridExtra)

#-----------#
#-Functions-#
#-----------#

#Function to create dual mesh (original code from Krainski et al. 2018)
source("rDualMesh.R")

#Logit function
logit <- function(pp)
{
  log(pp) - log(1-pp)
}

#Inverse logit
expit <- function(eta)
{
  1/(1+exp(-eta))
}

#-------------------------#
#-Spatio-temporal domains-#
#-------------------------#

#Full spatio-temporal extent
win <- owin(c(0, 3), c(0, 3)) #Window
loc.d <- 3 * cbind(c(0, 1, 1, 0, 0), c(0, 0, 1, 1, 0)) #Coordinates

#Number of pixels
npix <- 300
spatstat.options(npixel = npix)

#Create triangulated mesh over domain
mesh <- inla.mesh.2d(loc.domain = loc.d,
                     offset = c(0.3, 1),
                     max.edge = c(0.3, 0.7),
                     cutoff = 0.05)
```

```
#X range of mesh
x0 <- seq(min(mesh$loc[, 1]), max(mesh$loc[, 1]), length = npix)

#Y range of mesh
y0 <- seq(min(mesh$loc[, 2]), max(mesh$loc[, 2]), length = npix)

#Mesh window
Mwin <- owin(c(min(x0), max(x0)), c(min(y0), max(y0)))

#---------------#
#-Simulate LGCP-#
#---------------#

#Parameters of intensity function
beta <- c(6, NA)
beta[2] <- runif(1,0.5,1.5)

#Change in intensity
delta <- 1

#Parameters of thinning function
alpha <- c(logit(0.01), 0.85)

#Ecological covariate
RandomFields::RFoptions(spConform=FALSE)
xcov1 <- RFsimulate(model = RMgauss(scale = 1.25), x = x0, y = y0, grid =
TRUE)
xcov2 <- xcov1 + RFsimulate(model = RMgauss(var = 0.5, scale = 9.5), x = x0,
y = y0, grid = TRUE)
x1 <- (xcov1 - mean(c(xcov1, xcov2))/sd(c(xcov1, xcov2)))
x2 <- (xcov2 - mean(c(xcov1, xcov2))/sd(c(xcov1, xcov2)))

#Thinning covariate
pcov <- RFsimulate(model = RMgauss(scale = 1), x = x0, y = y0, grid = TRUE)
pcov <- (pcov - min(pcov))/sd(pcov)

#Range of Matern covaraince
range <- 1.2
#Scale of Matern covariance
kappa <- sqrt(8)/range
#Variance of Matern covaraince
sigma2 <- 0.2
#Precision of Matern covariance
tau <- sqrt(1/(sigma2*4*pi*kappa*kappa)
#Smoothness of Matern covaraince
nu <- 1

#Simulate spatial covariance
```

```
zcov <- RFsimulate(model = RMmatern(var = sigma2, scale = 1/kappa, nu = nu),
x = x0, y = y0, grid = TRUE)

#Intensity function
X1 <- as.im(x1, W = Mwin)[win]
X2 <- as.im(x2, W = Mwin)[win]
z <- as.im(zcov, W = Mwin)[win]

lambda1 <- exp(beta[1] + beta[2] * X1 + z)
lambda2 <- exp(beta[1] + delta[1] + beta[2] * X2 + z)

#Format intensity function
lambda1 <- as.im(lambda1, W=win)
lambda2 <- as.im(lambda2, W=win)

#Simulate LGCP
PP1 <- rpoispp(lambda1)[win]
PP2 <- rpoispp(lambda2)[win]

#---------------------------------#
#-Simulate opportunistic sampling-#
#---------------------------------#

#Thinning function
thin <- alpha[1] + alpha[2] * pcov
thin <- expit(thin)

#Format thinning function
thin <- as.im(thin, W=Mwin)[win]

#Latent observations
Z1 <- rbinom(n = PP1$n, size = 1, prob = thin[PP1])
PO1 <- PP1[Z1==1]

Z2 <- rbinom(n = PP2$n, size = 1, prob = thin[PP2])
PO2 <- PP2[Z2==1]

#-----------------#
#-Simulate counts-#
#-----------------#

#Number of counts
ncount <- 100

#Count locations
u.loc <- expand.grid(seq(0.15, 2.85, 0.3), seq(0.15, 2.85, 0.3))

#Latent abundance
N <- NULL
```

```
#Counts
count <- rep(0, ncount)

#Covariate value @ location
Cov <- NULL

#Unit area
unit.A <- NULL

for(j in 1:ncount){
  unit <- disc(radius = 0.15, centre = as.numeric(u.loc[j,]))
  count[j] <- PP1[unit]$n
  Cov[j] <- mean(as.im(x1, W=Mwin)[unit])
  unit.A[j] <- area(unit)
}

#Count dataframe
countdf <- data.frame(count, Cov, unit.A, u.loc)

#---------------#
#-SPDE approach-#
#---------------#

#Update mesh
mesh <- inla.mesh.2d(boundary = loc.d, max.edge = 0.3, cutoff = 0.05)

#Create SPDE objects (Matern covariance)
spde <- inla.spde2.matern(mesh)

#Create dual mesh for weights
dmesh <- book.mesh.dual(mesh)

#Format location domain
domain.polys <- Polygons(list(Polygon(loc.d)), '0')
domainSP <- SpatialPolygons(list(domain.polys))

#Weights of each node (area of dual mesh)
weight <- sapply(1:length(dmesh), function(i) {
  if (gIntersects(dmesh[i, ], domainSP))
    return(gArea(gIntersection(dmesh[i, ], domainSP)))
  else return(0)
})

rm(dmesh)

#--------------#
#-Compile Data-#
#--------------#
```

```
#Node index
nodes <- mesh$n

#Observation index
nobs <- sum(PO1$n, PO2$n)
ncount <- dim(countdf)[1]

#Period index
t_i <- rep(0:1, c(PO1$n, PO2$n))
t_n <- 2

#Counts
counts <- countdf$count

#Area @ each count location
Area <- countdf$unit.A

#Covaraites @ nodes
nloc <- as.ppp(mesh$loc[,1:2], W = Mwin)
nCov <- matrix(NA, nrow = nodes, ncol = t_n)
nCov[,1] <- as.im(x1, W = Mwin)[nloc]
nCov[,2] <- as.im(x2, W = Mwin)[nloc]
nBias <- as.im(pcov, W = Mwin)[nloc]

#Covariates @ obs
Cov <- c(as.im(x1, W = Mwin)[PO1],
         as.im(x2, W = Mwin)[PO2],
         countdf$Cov)

Bias <- c(as.im(pcov, W = Mwin)[PO1],
          as.im(pcov, W = Mwin)[PO2])

#Location of observations
locxy <- rbind(cbind(PO1$x, PO1$y)[,2:1],
               cbind(PO2$x, PO2$y)[,2:1],
               as.matrix(countdf[,4:5]))

#Projection matrix of observations
A <- as(inla.spde.make.A(mesh, locxy), "dgTMatrix")

#------------#
#-Prediction-#
#------------#

Grid <- as.matrix(expand.grid(seq(0.01,2.99,0.01), seq(0.01,2.99,0.01)))
npred <- dim(Grid)[1]
Apred <- as(inla.spde.make.A(mesh, Grid), "dgTMatrix")
predX <- matrix(NA, ncol = t_n, nrow = npred)
predX[,1] <- interp.im(X1, x = Grid[,1], y = Grid[,2])
```

```
predX[,2] <- interp.im(X2, x = Grid[,1], y = Grid[,2])

#------------------#
#-Integrated Model-#
#------------------#

#Compile TMB code (only once)
compile("Simulation.cpp")

#Load TMB code
dyn.load(dynlib("Simulation"))

#Compile data
data <- list("nodes" = nodes, "nobs" = nobs, "ncount" = ncount,
             "t_i" = t_i, "t_n" = t_n,
             "weight" = weight, "area" = Area, "A" = A, "counts" = counts,
             "nBias" = nBias, "nCov" = nCov, "Bias" = Bias, "Cov" = Cov,
             "spde" = spde$param.inla[c("M0","M1","M2")],
             "npred" = npred, "Apred" = Apred, "predX" = predX)

#PHASE 1: Fit fixed effects
#Parameters to estimate
params <- list("beta0" = 0, "beta1" = 1, "delta1" = 1,
               "alpha0" = logit(0.01), "alpha1" = 0,
               "log_kappa" = log(kappa), "log_tau_O" = 1,
               "omega" = rep(0, mesh$n))

#Define random effects
random = c("omega")

#Map
map <- list(
  "log_kappa" = as.factor(NA),
  "log_tau_O" = as.factor(NA),
  "omega" = factor(rep(NA, mesh$n)))

#Make AD objective function
obj1 <- MakeADFun(data = data, parameters = params, random = random, map =
map, DLL="Simulation")

#Trace parameters
obj1$env$tracepar <- TRUE

#Minimize objective function
opt1 <- nlminb(obj1$par, obj1$fn, obj1$gr)

#Calculate standard deviations
out1 <- sdreport(obj1)
```

```
#PHASE 2: Fit random effects
params <- list("beta0" = opt1$par[1], "beta1" = opt1$par[2], "delta1" =
opt1$par[3],
               "alpha0" = opt1$par[4], "alpha1" = opt1$par[5],
               "log_kappa" = log(kappa), "log_tau_O" = 1,
               "omega" = rep(0, mesh$n))

#Map
map <- list(
  "beta0" = as.factor(NA),
  "beta1" = as.factor(NA),
  "delta1" = as.factor(NA),
  "alpha0" = as.factor(NA),
  "alpha1" = as.factor(NA))

#Make AD objective function
obj2 <- MakeADFun(data = data, parameters = params, random = random, map =
map, DLL="Simulation")

#Trace parameters
obj2$env$tracepar <- TRUE

#Minimize objective function
opt2 <- nlminb(obj2$par, obj2$fn, obj2$gr)

#Calculate standard deviations
out2 <- sdreport(obj2)

#Store output

output <- as.data.frame(round(cbind(Truth = c(beta, delta, alpha, PP1$n,
PP2$n, kappa, tau), rbind(summary(out1)[!grepl("omega|sigma_O
|\\bkappa\\b|\\btau_O\\b",rownames(summary(out1))),],
summary(out2)[grepl("\\bkappa\\b|\\btau_O\\b", rownames(summary(out2))),])),
digits = 2))
colnames(output)[2:3] <- c("IM Estimate", "IM Std. Error")

#---------------------#
#-Presence-only model-#
#---------------------#

#Compile TMB code (only once)
compile("Simulation_PO.cpp")

#Load TMB code
dyn.load(dynlib("Simulation_PO"))

#Compile data
data <- list("nodes" = nodes, "nobs" = nobs, "t_i" = t_i, "t_n" = t_n,
```

```
              "weight" = weight, "A" = A[1:nobs,],
              "nBias" = nBias, "nCov" = nCov, "Bias" = Bias, "Cov" =
Cov[1:nobs],
              "spde" = spde$param.inla[c("M0","M1","M2")],
              "npred" = npred, "Apred" = Apred, "predX" = predX)

#PHASE 1: Fit fixed effects
#Parameters to estimate
params <- list("beta0" = 0, "beta1" = 1, "delta1" = 1,
               "alpha0" = logit(0.01), "alpha1" = 0,
               "log_kappa" = log(kappa), "log_tau_O" = 1,
               "omega" = rep(0, mesh$n))

#Map
map <- list(
  "log_kappa" = as.factor(NA),
  "log_tau_O" = as.factor(NA),
  "omega" = factor(rep(NA, mesh$n)))

#Make AD objective function
obj3 <- MakeADFun(data = data, parameters = params, random = random, map =
map, DLL="Simulation_PO")

#Trace parameters
obj3$env$tracepar <- TRUE

#Minimize objective function
opt3 <- nlminb(obj3$par, obj3$fn, obj3$gr)

#Calculate standard deviations
out3 <- sdreport(obj3)

params <- list("beta0" = opt1$par[1], "beta1" = opt1$par[2], "delta1" =
opt1$par[3],
               "alpha0" = opt1$par[4], "alpha1" = opt1$par[5],
               "log_kappa" = log(kappa), "log_tau_O" = 1,
               "omega" = rep(0, mesh$n))

#Map
map <- list(
  "beta0" = as.factor(NA),
  "beta1" = as.factor(NA),
  "delta1" = as.factor(NA),
  "alpha0" = as.factor(NA),
  "alpha1" = as.factor(NA))

#Define random effects
random = c("omega")
```

```
#Make AD objective function
obj4 <- MakeADFun(data = data, parameters = params, map = map, random =
random, DLL="Simulation_PO")

#Trace parameters
obj4$env$tracepar <- TRUE

#Minimize objective function
opt4 <- nlminb(obj4$par, obj4$fn, obj4$gr)

#Calculate standard deviations
out4 <- sdreport(obj4)

#Store output

output <- merge(output,

as.data.frame(round(rbind(summary(out3)[!grepl("omega|sigma_O|\\bkappa\\b|\\b
tau_O\\b", rownames(summary(out3))),],
summary(out4)[grepl("\\bkappa\\b|\\btau_O\\b", rownames(summary(out4))),]),
digits = 2)), by='row.names', all=TRUE)

colnames(output)[5:6] <- c("PO Estimate", "PO Std. Error")
output$Row.names[7:8] <- c("N[1]", "N[2]")
rownames(output) <- output$Row.names
output <- output[,-1]

convergence <- data.frame("pdHess" = c(out1$pdHess, out2$pdHess, out3$pdHess,
out4$pdHess),
            "grd.size" = c(all(abs(out1$gradient.fixed) < 0.01),
                           all(abs(out2$gradient.fixed) < 0.01),
                           all(abs(out3$gradient.fixed) < 0.01),
                           all(abs(out4$gradient.fixed) < 0.01)))

rownames(convergence) <- c("IM_fixed", "IM_random", "PO_fixed", "PO_random")

out <- list(output, convergence)

message("finished")

#-----------#
#-Save file-#
#-----------#

ID <- length(list.files("./Output/")) + 1
save(out, file = paste("./Output/output", ID, ".Rds", sep=""))
```

Template Model Builder C++ code for the simulation study

```
#include <TMB.hpp>
// Integrated Log Gaussian Cox Process
template <class Type>
Type objective_function<Type>::operator() ()
{

  // objective function -- joint negative log-likelihood
  using namespace R_inla;
  using namespace density;
  using namespace Eigen;

  // DATA //

  // Indices
  DATA_INTEGER( nodes );  // Number of nodes
  DATA_INTEGER( nobs );   // Number of presence-only observations
  DATA_INTEGER( ncount ); // Number of structured sampling sites
  DATA_INTEGER( t_n );    // Number of stages
  DATA_IVECTOR( t_i );    // Stage ID for data

  // Projection & weight data
  DATA_VECTOR( weight );   // Node weight
  DATA_VECTOR( area );     // Site area for counts
  DATA_SPARSE_MATRIX( A ); // Projection matrix

  // Count dataset
  DATA_VECTOR( counts ); // Number of counts per site

  // Covaraite dataset
  DATA_VECTOR( nBias ); //Sampling bias @ nodes
  DATA_MATRIX( nCov );  //Environmental covariate @ nodes

  DATA_VECTOR( Bias ); //Sampling bias @ observations
  DATA_VECTOR( Cov ); //Environmental covariate @ data

  // SPDE objects
  DATA_STRUCT(spde,spde_t); // Sparse matrix for Matern covariance structure

  // Prediction
  DATA_INTEGER( npred );        // Number of pixels for prediction
  DATA_MATRIX( predX );         // Environmental covariate for prediction
  DATA_SPARSE_MATRIX( Apred );  // Projection matrix for prediction

  // PARAMETERS //

  // Fixed effects
  PARAMETER( beta0 );     // Baseline population density
```

```
  PARAMETER( beta1 );     // Effect of environmental covariate
  PARAMETER( delta1 );    // Effect of stage (population change)
  PARAMETER( alpha0 );    // Thinning function intercept
  PARAMETER( alpha1 );    // Effect of sampling bias
  PARAMETER( log_kappa ); // Scale parameter of Matern covariance
  PARAMETER( log_tau_O ); // Precision parameter of Matern covariance

  // Random effects
  PARAMETER_VECTOR( omega ); // Spatial random effect

  // Population density at each stage
  vector<Type> beta(t_n);
  beta(0) = beta0;
  beta(1) = beta0 + delta1;

  // Derived parameters
  Type kappa = exp(log_kappa);
  Type tau_O = exp(log_tau_O);
  Type range = sqrt(8)/kappa;
  Type sigma_O = 1/sqrt(4*PI*tau_O*tau_O*kappa*kappa);

  vector<Type> jnll_comp(4);
  jnll_comp.setZero();

  // Probability of random effects
  SparseMatrix<Type> Q = Q_spde(spde,kappa); // Matern covariance (see R_inla
namespace)
  jnll_comp(0) += GMRF(Q)( omega );

  // Holding values
  vector<Type> Omega(nodes);
  vector<Type> omg(nobs + ncount);

  // Transform GMRFs
  for(int k=0; k<nodes; k++){
    Omega(k) = omega(k) / tau_O;
  }

  // Project GMRFs
  omg = A * Omega; // Project omega to points

  // Intensity function @ nodes
  for(int t=0; t<t_n; t++){
    for(int k=0; k<nodes; k++){
      jnll_comp(1) += weight(k) * exp(alpha0 + alpha1 * nBias(k) + beta(t) +
beta1 * nCov(k,t)+ Omega(k)) / (exp(alpha0 + alpha1 * nBias(k)) + 1); //
Integration nodes
    }
  }
```

```
  // Intensity function @ presence-only data
  for(int i=0; i<nobs; i++){
    jnll_comp(2) -= alpha0 + alpha1 * Bias(i) + beta(t_i(i)) + beta1 * Cov(i)
+ omg(i) - log(exp(alpha0 + alpha1 * Bias(i)) + 1); // Observation points
  }

  // Intensity function @ counts
  vector<Type> lambda(ncount);
  for(int j=0; j<ncount; j++){
    lambda(j) = area(j) * exp(beta0 + beta1 * Cov(j+nobs) + omg(j+nobs));
    jnll_comp(3) -= dpois(counts(j), lambda(j), true);
  }

  // Prediction
  vector<Type> predO(npred);
  predO = Apred * Omega;

  vector<Type> pred(t_n);
  vector<Type> Npred(t_n);

  for(int t=0; t<t_n; t++){
    for(int g=0; g<npred; g++){
      pred(t) += exp(beta(t) + beta1 * predX(g,t) + predO(g));
    }
    Npred(t) = pred(t)/npred * 9;
  }


  // Joint NLL
  Type jnll = jnll_comp.sum();

  // Reporting
  REPORT( kappa );
  REPORT( tau_O );
  REPORT( sigma_O );
  REPORT( range );
  REPORT( Npred );
  ADREPORT( Npred );
  ADREPORT( kappa );
  ADREPORT( tau_O );
  ADREPORT( sigma_O );

  REPORT( jnll_comp );
  REPORT( jnll );

  return jnll;
}
```

Supplementary S3: Case study details, code, and output

This supplement contains detailed information on the covariates used in the case study along with additional results. We also include the R and C++ code to run the monarch analysis. Please also refer to archived repositories on either Zenodo (DOI: 10.5281/zenodo.8433370) or GitHub (https://github.com/zipkinlab/Farr_etal_2024_MEE).

Covariate descriptions

We acquired normalized difference vegetation index (NDVI) values from Terra Moderate Resolution Imaging Spectroradiometer (MODIS v.006) Vegetation Indices (MOD13A2) at a spatial resolution of 1 km^2 and a temporal resolution of 16 days (Didan 2015). At each presence-only location, s_i, and at each single-visit site, D_j, we extracted the mean NDVI value within a 10-km radius circle centered on each location during the appropriate 16-day window.

We used the number of growing degree days (GDD) to describe the thermal conditions across the spring migratory pathway. GDD is the accumulation of heat within a physiologically relevant range of temperatures that allows for development (McMaster & Wilhelm 1997). We approximated GDD for a given location as:

$$GDD = \sum_{n=1}^{N} \begin{cases} T_{min} & if\ \bar{T}_n < T_{min} \\ \bar{T}_n - T_{min} & if\ T_{min} \leq \bar{T}_n \leq T_{max} \\ T_{max} & if\ T_{max} < \bar{T}_n \end{cases}$$

where $\bar{T}_n$ is the average temperature of a day n, T_{min} is the minimum temperature threshold (i.e., 11.5°C for monarchs), T_{max} is the maximum temperature threshold (i.e., 33°C for monarchs), and N is the number of days accumulated since $n = 1$. Here, we summarize GDD across 2 weeks, $N = 14$, and set $n = 1$ to be 14 days prior to each observation or survey (i.e., GDD is summarized across the 2 weeks directly before an observation or survey). Average daily temperatures were obtained from Daymet (Thornton et al. 2020), which interpolates temperature values across North America at a 1 km^2 resolution.

We used two covariates to correct for sampling biases. We summed the number of non-monarch butterfly observations in the superfamily *Papilionoidea* from iNaturalist within a 10 km buffer for each observation or survey. We also used human population density (CIESIN 2018) to correct for sampling bias as local human population densities are likely to be positively correlated with local numbers of observers (Geldmann et al. 2016). We extracted population density values within a 10 km buffer for each observation or survey.

Additional model results

Table S3.1. Estimates (mean and 95% confidence intervals) of monarch density (adult monarchs per 100-m^2) for each year and period.

Year	Period	Density (adults per 100 m^2)
2016	Early Spring	0.001 (CI 0.001, 0.002)
2016	Late Spring	0.002 (CI 0.001, 0.002)
2016	Early Summer	0.003 (CI 0.003, 0.004)
2017	Early Spring	0.004 (CI 0.003, 0.005)
2017	Late Spring	0.004 (CI 0.004, 0.006)
2017	Early Summer	0.009 (CI 0.008, 0.011)
2018	Early Spring	0.010 (CI 0.008, 0.012)
2018	Late Spring	0.012 (CI 0.010, 0.015)
2018	Early Summer	0.022 (CI 0.020, 0.025)

Table S3.2. Estimates (mean and 95% confidence intervals) of the effect of NDVI and GDD on each period on monarch population density, reported on the log-scale.

Covariate	Period	Effect (log-scale)
$NDVI$	Early Spring	0.22 (CI 0.09, 0.36)
$NDVI$	Late Spring	1.87 (CI 1.63, 2.14)
$NDVI$	Early Summer	0.41 (CI 0.30, 0.52)
$NDVI^2$	Early Spring	0.23 (CI 0.16, 0.29)
$NDVI^2$	Late Spring	-1.62 (CI -1.87, -1.40)
$NDVI^2$	Early Summer	-0.33 (CI -0.42, -0.24)
GDD	Early Spring	0.97 (CI 0.73, 1.23)
GDD	Late Spring	0.56 (CI 0.31, 0.82)
GDD	Early Summer	0.02 (CI -0.13, 0.17)
GDD^2	Early Spring	-1.17 (CI -0.13, 0.17)
GDD^2	Late Spring	-0.89 (CI -0.94, -0.67)
GDD^2	Early Summer	-0.09 (CI -0.21, 0.02)

Case study script

Below is the R script to run the case study analysis on monarch butterflies.

```
#-----------#
#-Libraries-#
#-----------#

library(tidyverse)
library(sf)
library(INLA)
library(TMB)

#Function to create dual mesh (original code from Krainski et al. 2018)
source("~/Monarchs/DataAnalysis/rDualMesh.R")

#-----------#
#-Load data-#
#-----------#

#Spring domain
domain_sp <-
st_read("~/Monarchs/DataFormatting/BaseData/Domain/Domain_Spring.shp")

#Summer domain
domain_su <-
st_read("~/Monarchs/DataFormatting/BaseData/Domain/Domain_Summer.shp")

#Monarch data
load(file = "~/Monarchs/DataFormatting/FormattedData.Rdata")

#Mesh node data
load(file = '~/Monarchs/DataFormatting/FormattedNodeData.Rdata')

#-------------#
#-Format data-#
#-------------#

#Geographic projection
prj <- "+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=37.5 +lon_0=-96 +x_0=0
+y_0=0 +ellps=GRS80 +datum=NAD83 +units=km +no_defs"

#Set projection
domain_sp <- st_transform(domain_sp, crs = st_crs(prj))
domain_su <- st_transform(domain_su, crs = st_crs(prj))

#Spring domain boundary
domain_sp_seg <- inla.sp2segment(as(domain_sp, "Spatial"))
```

```
#Summer domain boundary
domain_su_seg <- inla.sp2segment(as(domain_su, "Spatial"))

#Construct triangulated mesh
mesh <- list()

mesh[[1]] <- inla.mesh.2d(boundary=domain_sp_seg,
                          max.edge = 75,
                          cutoff=30)

mesh[[2]] <- inla.mesh.2d(boundary=domain_sp_seg,
                          max.edge = 75,
                          cutoff=30)

mesh[[3]] <- inla.mesh.2d(boundary=domain_su_seg,
                          max.edge = 75,
                          cutoff = 30)

#Create SPDE objects (Matern covariance structure)
spde1 <- inla.spde2.matern(mesh[[1]])
spde2 <- inla.spde2.matern(mesh[[2]])
spde3 <- inla.spde2.matern(mesh[[3]])

#Create dual mesh for weights (see Krainski et al. 2018)
dmesh1 <- book.mesh.dual(mesh[[1]])
dmesh1 <- st_as_sf(dmesh1)
st_crs(dmesh1) <- st_crs(domain_sp)

dmesh2 <- book.mesh.dual(mesh[[2]])
dmesh2 <- st_as_sf(dmesh2)
st_crs(dmesh2) <- st_crs(domain_sp)

dmesh3 <- book.mesh.dual(mesh[[3]])
dmesh3 <- st_as_sf(dmesh3)
st_crs(dmesh3) <- st_crs(domain_su)

#Weights of each node (area of dual mesh)
weight1 <- sapply(1:dim(dmesh1)[1], function(i) {
  if (st_intersects(dmesh1[i,], domain_sp, sparse = FALSE))
    return(st_area(st_intersection(dmesh1[i,], domain_sp)))
  else return(0)
})

weight2 <- sapply(1:dim(dmesh2)[1], function(i) {
  if (st_intersects(dmesh2[i,], domain_sp, sparse = FALSE))
    return(st_area(st_intersection(dmesh2[i,], domain_sp)))
  else return(0)
})
```

```
weight3 <- sapply(1:dim(dmesh3)[1], function(i) {
  if (st_intersects(dmesh3[i,], domain_su, sparse = FALSE))
    return(st_area(st_intersection(dmesh3[i,], domain_su)))
  else return(0)
})

#Projection matrix of observations to nodes
A1 <- as(inla.spde.make.A(mesh[[1]], st_coordinates(Data %>% filter(period ==
1))), "dgTMatrix")
A2 <- as(inla.spde.make.A(mesh[[2]], st_coordinates(Data %>% filter(period ==
2))), "dgTMatrix")
A3 <- as(inla.spde.make.A(mesh[[3]], st_coordinates(Data %>% filter(period ==
3))), "dgTMatrix")

#--------------#
#-Compile data-#
#--------------#

#Number of years
t_n <- Data %>% summarize(out = n_distinct(yr)) %>% select(out) %>% .$out

#Number of stages
p_n <- Data %>% summarize(out = n_distinct(period)) %>% select(out) %>% .$out

#Number of nodes in early spring
nodes1 <- mesh[[1]]$n

#Number of nodes in late spring
nodes2 <- mesh[[2]]$n

#Number of nodes in early summer
nodes3 <- mesh[[3]]$n

#Number of nodes across stages and years
nodes <- as.integer(t_n*(nodes1 + nodes2 + nodes3))

#MOVE TO NODE FORMATTING
NodeDF$period <- as.factor(rep(1:3, c(nodes1 * 3, nodes2 * 3, nodes3 * 3)))

#Year-stage identifier for nodes
tp_k <- as.integer(fct_cross(NodeDF$period, NodeDF$Year)) - 1

#Stage identifier for nodes
p_k <- as.integer(NodeDF$period) - 1

#Number of presence-only observations
nobs <- dim(Data %>% filter(type == "obs"))[1]

#Number of presence-only observations in early spring
```

```
nobs1 <- dim(Data %>% filter(type == "obs" & period == 1))[1]

#Number of presence-only observations in late spring
nobs2 <- dim(Data %>% filter(type == "obs" & period == 2))[1]

#Number of presence-only observations in early summer
nobs3 <- dim(Data %>% filter(type == "obs" & period == 3))[1]

#Number of single-visit sites
ncount <- dim(Data %>% filter(type == "count"))[1]

#Number of single-visit sites in early spring
ncount1 <- dim(Data %>% filter(type == "count" & period == 1))[1]

#Number of single-visit sites in late spring
ncount2 <- dim(Data %>% filter(type == "count" & period == 2))[1]

#Number of single-visit sites in early summer
ncount3 <- dim(Data %>% filter(type == "count" & period == 3))[1]

#Single-visit counts @ each site
counts <- as.numeric(Data %>% filter(type == "count") %>% select(count) %>%
.$count)

#Year-stage identifier for data
tp_i <- as.integer(fct_cross(as.factor(Data$period), as.factor(Data$yr))) - 1

#Stage identifier for data
p_i <- as.integer(as.factor(Data$period)) - 1

#Weight of each node @ 100 m^2
weight <- rep(c(weight1, weight2, weight3), t_n) * 1e4

#Area/effort offset
area <- as.numeric(Data %>% filter(type == "count") %>% select(effort) %>%
.$effort)

#Covert to 100 m^2
area <- area*6*1000*1e-2

#Observation covariates
Data <- Data %>% mutate(daysaccum = as.numeric(as.Date(paste0(yr, "-",
mo_day)) -
                                                  as.Date(paste0(yr,
ifelse(period == 1, "-03-01",

ifelse(period == 2, "-03-22", "-04-26"))))) + 1,
                        gdd.avg = gdd2/daysaccum)
```

```
#Normalized Difference Vegetation Index @ data
NDVI <- Data$NDVI

#Averaged daily growing degree days @ data
GDD <- Data$gdd.avg

#Number of other butterfly observations @ data
Bias <- as.numeric(Data %>% filter(type == "obs") %>% select(Bias) %>%
.$Bias)

#Population density @ data
PopD <- as.numeric(Data %>% filter(type == "obs") %>% select(PopD) %>%
.$PopD)


#Node covariates
NodeDF <- NodeDF %>% mutate(daysaccum = ifelse(period == 1, 35, 42),
                            gdd.avg = gdd2/daysaccum)

#Normalized Difference Vegetation Index @ nodes
nNDVI <- NodeDF$NDVI

#Averaged daily growing degree days @ nodes
nGDD <- NodeDF$gdd.avg

#Number of other butterfly observations @ nodes
nBias <- NodeDF$Bias

#Population density @ nodes
nPopD <- NodeDF$PopD

#Scale covariates for both data & nodes
ndvi.stan.data <- (NDVI - mean(c(NDVI, nNDVI)))/sd(c(NDVI, nNDVI))
gdd.stan.data <- (GDD - mean(c(GDD, nGDD)))/sd(c(GDD, nGDD))
bias.stan.data <- (Bias - mean(c(Bias, nBias)))/sd(c(Bias, nBias))
pop.stan.data <- (PopD - mean(c(PopD, nPopD)))/sd(c(PopD, nPopD))

ndvi.stan.nodes <- (nNDVI - mean(c(NDVI, nNDVI)))/sd(c(NDVI, nNDVI))
gdd.stan.nodes <- (nGDD - mean(c(GDD, nGDD)))/sd(c(GDD, nGDD))
bias.stan.nodes <- (nBias - mean(c(Bias, nBias)))/sd(c(Bias, nBias))
pop.stan.nodes <- (nPopD - mean(c(PopD, nPopD)))/sd(c(PopD, nPopD))

NDVI <- ndvi.stan.data
GDD <- gdd.stan.data
Bias <- bias.stan.data
PopD <- pop.stan.data

nNDVI <- ndvi.stan.nodes
nGDD <- gdd.stan.nodes
```

```
nBias <- bias.stan.nodes
nPopD <- pop.stan.nodes

#--------------#
#-Optimize nll-#
#--------------#

#Compile TMB code (only once)
# compile("./DataAnalysis/IntegratedModel.cpp")

#Load TMB code
dyn.load(dynlib("./DataAnalysis/IntegratedModel"))

#PHASE 1: Fit fixed effects

#Compile data for TMB
data <- list("nodes1" = nodes1, "nodes2" = nodes2, "nodes3" = nodes3,
             "nobs1" = nobs1, "nobs2" = nobs2, "nobs3" = nobs3,
             "ncount" = ncount, "ncount1" = ncount1, "ncount2" = ncount2,
"ncount3" = ncount3,
             "counts" = counts,
             "t_n" = t_n, "p_n" = p_n, "tp_k" = tp_k, "tp_i" = tp_i, "p_i" =
p_i, "p_k" = p_k,
             "weight" = weight, "area" = area,
             "A1" = A1, "spde1" = spde1$param.inla[c("M0","M1","M2")],
             "A2" = A2, "spde2" = spde2$param.inla[c("M0","M1","M2")],
             "A3" = A3, "spde3" = spde3$param.inla[c("M0","M1","M2")],
             "nBias" = nBias, "nPopD" = nPopD,
             "nNDVI" = nNDVI, "nGDD" = nGDD,
             "Bias" = Bias, "PopD" = PopD,
             "NDVI" = NDVI, "GDD" = GDD)

#Parameters to estimate
params <- list("beta0" = 0,
               "beta1" = rep(0,3), "beta2" = rep(0,3),
               "beta3" = rep(0,3), "beta4" = rep(0,3),
               "delta1" = 0, "delta2" = 0,
               "gamma1" = 0, "gamma2" = 0,
               "alpha0" = 0, "alpha1" = rep(0,9), "alpha2" = rep(0,9),
               "log_kappa" = 0, "log_tau" = 1,
               "omega1" = rep(0, nodes1),
               "omega2" = rep(0, nodes2),
               "omega3" = rep(0, nodes3))

#Hold random effects constant
map <- list(
  "log_kappa" = as.factor(NA),
  "log_tau" = as.factor(NA),
  "omega1" = factor(rep(NA, nodes1)),
```

```
  "omega2" = factor(rep(NA, nodes2)),
  "omega3" = factor(rep(NA, nodes3))
)

#Random effects
random <- c("omega1", "omega2", "omega3")

#Make AD objective function
obj1 <- MakeADFun(data = data, parameters = params, random = random, map =
map, DLL="IntegratedModel_Final")

#Trace parameters
obj1$env$tracepar <- TRUE

#Minimize objective function
opt1 <- nlminb(obj1$par, obj1$fn, obj1$gr)

#AIC
2 * length(obj1$par) - 2 * (-1 * sum(obj1$report()$jnll_comp[3:12]))

#Output
out1 <- sdreport(obj1)
summary(out1)

#Extract parameter values
beta0 <- out1$par.fixed[1]
beta1 <- out1$par.fixed[2:4]
beta2 <- out1$par.fixed[5:7]
beta3 <- out1$par.fixed[8:10]
beta4 <- out1$par.fixed[11:13]
delta1 <- out1$par.fixed[14]
delta2 <- out1$par.fixed[15]
gamma1 <- out1$par.fixed[16]
gamma2 <- out1$par.fixed[17]

#Expected population densities per stage and year
beta <- NULL

#2016 stage 1 (early spring)
beta[1] <- beta0 + gamma1 + gamma2 + delta1 + delta2

#2016 stage 2 (late spring)
beta[2] <- beta0 + gamma1 + delta1 + delta2

#2016 stage 3 (early summer)
beta[3] <- beta0 + delta1 + delta2

#2017 stage 1 (early spring)
beta[4] <- beta0 + gamma1 + gamma2 + delta1
```

```
#2017 stage 2 (late spring)
beta[5] <- beta0 + gamma1 + delta1

#2017 stage 3 (early summer)
beta[6] <- beta0 + delta1

#2018 stage 1 (early spring)
beta[7] <- beta0 + gamma1 + gamma2

#2018 stage 2 (late spring)
beta[8] <- beta0 + gamma1

#2018 stage 3 (early summer)
beta[9] <- beta0

#Compile output
Output <- data.frame(year = factor(rep(2016:2018, each = 3)),
                     period = factor(rep(1:3, 3)),
                     mean.den = beta)
Output$lower.den <- NA
Output$upper.den <- NA

#Profile confidence intervals
Output[1,4:5] <- confint(tmbprofile(obj1, lincomb =
as.numeric(names(obj1$par) %in%
c("beta0","gamma1","gamma2","delta1","delta2"))))
Output[2,4:5] <- confint(tmbprofile(obj1, lincomb =
as.numeric(names(obj1$par) %in% c("beta0","gamma1","delta1","delta2"))))
Output[3,4:5] <- confint(tmbprofile(obj1, lincomb =
as.numeric(names(obj1$par) %in% c("beta0","delta1","delta2"))))
Output[4,4:5] <- confint(tmbprofile(obj1, lincomb =
as.numeric(names(obj1$par) %in% c("beta0","gamma1","gamma2","delta1"))))
Output[5,4:5] <- confint(tmbprofile(obj1, lincomb =
as.numeric(names(obj1$par) %in% c("beta0","gamma1","delta1"))))
Output[6,4:5] <- confint(tmbprofile(obj1, lincomb =
as.numeric(names(obj1$par) %in% c("beta0","delta1"))))
Output[7,4:5] <- confint(tmbprofile(obj1, lincomb =
as.numeric(names(obj1$par) %in% c("beta0","gamma1","gamma2"))))
Output[8,4:5] <- confint(tmbprofile(obj1, lincomb =
as.numeric(names(obj1$par) %in% c("beta0","gamma1"))))
Output[9,4:5] <- confint(tmbprofile(obj1, lincomb =
as.numeric(names(obj1$par) %in% c("beta0"))))

#PHASE 2: Fit random effects

#Parameters to estimate
params2 <- list("beta0" = beta0, "beta1" = beta1,
                "beta2" = beta2, "beta3" = beta3,
```

```
                "beta4" = beta4,
                "delta1" = delta1, "delta2" = delta2,
                "gamma1" = gamma1, "gamma2" = gamma2,
                "alpha0" = out1$par.fixed[18],
                "alpha1" = out1$par.fixed[19:27],
                "alpha2" = out1$par.fixed[28:36],
                "log_kappa" = 0, "log_tau" = 1,
                "omega1" = rep(0, nodes1), "omega2" = rep(0, nodes2),
"omega3" = rep(0, nodes3))

#Hold fixed effects constant
map2 <- list(
  "alpha0" = as.factor(NA),
  "alpha1" = factor(rep(NA, 9)),
  "alpha2" = factor(rep(NA, 9)),
  "beta0" = as.factor(NA),
  "beta1" = factor(rep(NA, 3)),
  "beta2" = factor(rep(NA, 3)),
  "beta3" = factor(rep(NA, 3)),
  "beta4" = factor(rep(NA, 3)),
  "delta1" =  as.factor(NA),
  "delta2" = as.factor(NA),
  "gamma1" = as.factor(NA),
  "gamma2" = as.factor(NA)
)

#Make AD objective function
obj2 <- MakeADFun(data = data, parameters = params2, map = map2, random =
random, DLL="IntegratedModel ")

#Trace parameters
obj2$env$tracepar <- TRUE

#Minimize objective function
opt2 <- nlminb(obj2$par, obj2$fn, obj2$gr)

#Output
out2 <- sdreport(obj2)

#Estimated parameter values
Effect <- data.frame(param = rep(c("beta1", "beta2", "beta3", "beta4"), each
= 3),
                     period = factor(rep(1:3, 4)),
                     mean.den = c(beta1, beta2, beta3, beta4))

Effect$lower <- NA
Effect$upper <- NA

Effect[1,4:5] <- confint(tmbprofile(obj1, lincomb = c(0,1,rep(0,34))))
```

```
Effect[2,4:5] <- confint(tmbprofile(obj1, lincomb = c(0,0,1,rep(0,33))))
Effect[3,4:5] <- confint(tmbprofile(obj1, lincomb = c(0,0,0,1,rep(0,32))))
Effect[4,4:5] <- confint(tmbprofile(obj1, lincomb = c(0,0,0,0,1,rep(0,31))))
Effect[5,4:5] <- confint(tmbprofile(obj1, lincomb =
c(0,0,0,0,0,1,rep(0,30))))
Effect[6,4:5] <- confint(tmbprofile(obj1, lincomb = c(rep(0,6),1,rep(0,29))))
Effect[7,4:5] <- confint(tmbprofile(obj1, lincomb = c(rep(0,7),1,rep(0,28))))
Effect[8,4:5] <- confint(tmbprofile(obj1, lincomb = c(rep(0,8),1,rep(0,27))))
Effect[9,4:5] <- confint(tmbprofile(obj1, lincomb = c(rep(0,9),1,rep(0,26))))
Effect[10,4:5] <- confint(tmbprofile(obj1, lincomb =
c(rep(0,10),1,rep(0,25))))
Effect[11,4:5] <- confint(tmbprofile(obj1, lincomb =
c(rep(0,11),1,rep(0,24))))
Effect[12,4:5] <- confint(tmbprofile(obj1, lincomb =
c(rep(0,12),1,rep(0,23))))
```

Template Model Builder code for case study

TMB code to generate the objective function containing the negative log likelihood for optimization.

```
#include <TMB.hpp>
// Integrated Log Gaussian Cox Process
template <class Type>
Type objective_function<Type>::operator() ()
{

  // Objective function -- joint negative log-likelihood
  using namespace R_inla;
  using namespace density;
  using namespace Eigen;

  // Indices
  DATA_INTEGER( nodes1 );  // Number of nodes in early spring
  DATA_INTEGER( nodes2 );  // Number of nodes in late spring
  DATA_INTEGER( nodes3 );  // Number of nodes in early summer
  DATA_INTEGER( nobs1 );   // Number of presence-only observations in early
spring
  DATA_INTEGER( nobs2 );   // Number of presence-only observations in late
spring
  DATA_INTEGER( nobs3 );   // Number of presence-only observations in early
summer
  DATA_INTEGER( ncount );  // Number of structured sampling sites
  DATA_INTEGER( ncount1 ); // Number of structured sampling sites in early
spring
  DATA_INTEGER( ncount2 ); // Number of structured sampling sites in late
spring
  DATA_INTEGER( ncount3 ); // Number of structured sampling sites in early
summer
  DATA_INTEGER( t_n );     // Number of years
  DATA_INTEGER( p_n );     // Number of stages
  DATA_IVECTOR( tp_i );    // Index for year and stage combination for the
data
  DATA_IVECTOR( tp_k );    // Index for year and stage combination for the
nodes
  DATA_IVECTOR( p_i );     // Stage ID for the data
  DATA_IVECTOR( p_k );     // Stage ID for the nodes

  // Projection & weight data
  DATA_VECTOR( weight );    // Node weight
  DATA_VECTOR( area );      // Area sweept for counts
  DATA_SPARSE_MATRIX( A1 ); // Projection matrix early spring
  DATA_SPARSE_MATRIX( A2 ); // Projection matrix late spring
  DATA_SPARSE_MATRIX( A3 ); // Projection matrix early summer
```

```
  //Count data
  DATA_VECTOR( counts ); // Number of counts per site

  // Covaraite dataset
  DATA_VECTOR( nBias ); // Number of other butterfly observations @ nodes
  DATA_VECTOR( nPopD ); // Human population density @ nodes
  DATA_VECTOR( nNDVI ); // NDVI @ nodes
  DATA_VECTOR( nGDD );  // GDD @ nodes

  DATA_VECTOR( Bias ); // Number of other butterfly observations @ data
  DATA_VECTOR( PopD ); // Human population density @ data
  DATA_VECTOR( NDVI ); // NDVI @ data
  DATA_VECTOR( GDD );  // GDD @ data

  // SPDE objects
  DATA_STRUCT(spde1,spde_t); // Sparse matrix for Matern covariance structure
in early spring
  DATA_STRUCT(spde2,spde_t); // Sparse matrix for Matern covariance structure
in late spring
  DATA_STRUCT(spde3,spde_t); // Sparse matrix for Matern covariance structure
in early summer

  // Fixed effects
  PARAMETER( beta0 );         // Baseline population density
  PARAMETER_VECTOR( beta1 );  // Effect of NDVI
  PARAMETER_VECTOR( beta2 );  // Quadratic effect of NDVI
  PARAMETER_VECTOR( beta3 );  // Effect of GDD
  PARAMETER_VECTOR( beta4 );  // Quadratic effect of GDD
  PARAMETER( delta1 );        // Effect of 2016
  PARAMETER( delta2 );        // Effect of 2017
  PARAMETER( gamma1 );        // Effect of early spring
  PARAMETER( gamma2 );        // Effect of late spring
  PARAMETER( alpha0 );        // Thinning function intercept
  PARAMETER_VECTOR( alpha1 ); // Effect of number of other butterfly
observations
  PARAMETER_VECTOR( alpha2 ); // Effect of human population density
  PARAMETER( log_kappa );     // Scale parameter of Matern covariance
  PARAMETER( log_tau );       // Precision parameter of Matern covariance

  // Random effects
  PARAMETER_VECTOR( omega1 ); // Spatial random effect in early spring
  PARAMETER_VECTOR( omega2 ); // Spatial random effect in late spring
  PARAMETER_VECTOR( omega3 ); // Spatial random effect in early summer

  // Population density at each stage and year
  vector<Type> beta(t_n * p_n);
  beta(0) = beta0 + gamma1 + gamma2 + delta1 + delta2; //2016 early spring
  beta(1) = beta0 + gamma1 + delta1 + delta2;          //2016 late spring
  beta(2) = beta0 + delta1 + delta2;                   //2016 early summer
```

```
beta(3) = beta0 + gamma1 + gamma2 + delta1;          //2017 early spring
beta(4) = beta0 + gamma1 + delta1;                   //2017 late spring
beta(5) = beta0 + delta1;                            //2017 early summer
beta(6) = beta0 + gamma1 + gamma2;                   //2018 early spring
beta(7) = beta0 + gamma1;                            //2018 late spring
beta(8) = beta0;                                     //2018 early summer

// Derived parameters for computational purposes
Type kappa = exp(log_kappa);
Type tau = exp(log_tau);
Type range = sqrt(8)/kappa;
Type sigma = 1/sqrt(4*PI*tau*tau*kappa*kappa);

vector<Type> jnll_comp(12);
jnll_comp.setZero();

// Probability of random effects
SparseMatrix<Type> Q1 = Q_spde(spde1,kappa);
SparseMatrix<Type> Q2 = Q_spde(spde2,kappa);
SparseMatrix<Type> Q3 = Q_spde(spde3,kappa);
jnll_comp(0) += GMRF(Q1)( omega1 );
jnll_comp(1) += GMRF(Q2)( omega2 );
jnll_comp(2) += GMRF(Q3)( omega3 );

// Holding values
vector<Type> Omega1(nodes1);
vector<Type> Omega2(nodes2);
vector<Type> Omega3(nodes3);

vector<Type> omg1(nobs1 + ncount1);
vector<Type> omg2(nobs2 + ncount2);
vector<Type> omg3(nobs3 + ncount3);

// Transform GMRFs

//Early spring
for(int k=0; k<nodes1; k++){
  Omega1(k) = omega1(k) / tau;
}
//Late spring
for(int k=0; k<nodes2; k++){
  Omega2(k) = omega2(k) / tau;
}
//Summer
for(int k=0; k<nodes3; k++){
  Omega3(k) = omega3(k) / tau;
}

// Project GMRFs
```

```
  omg1 = A1 * Omega1;
  omg2 = A2 * Omega2;
  omg3 = A3 * Omega3;


  // Intensity function @ nodes
  for(int t=0, j=0, i=nodes1*t_n, g=(nodes1+nodes2)*t_n; t<t_n; t++){
    for(int k=0; k<nodes1; k++, j++){
      jnll_comp(3) += weight(j) * exp(alpha0 + alpha1(tp_k(j)) * nBias(j) +
alpha2(tp_k(j)) * nPopD(j) + beta(tp_k(j)) + beta1(p_k(j)) * nNDVI(j) +
beta2(p_k(j)) * nNDVI(j) * nNDVI(j) + beta3(p_k(j)) * nGDD(j) + beta4(p_k(j))
* nGDD(j) * nGDD(j) + Omega1(k)) / (exp(alpha0 + alpha1(tp_k(j)) * nBias(j) +
alpha2(tp_k(j)) * nPopD(j)) + 1);
    }
    for(int k=0; k<nodes2; k++, i++){
      jnll_comp(4) += weight(i) * exp(alpha0 + alpha1(tp_k(i)) * nBias(i) +
alpha2(tp_k(i)) * nPopD(i) + beta(tp_k(i)) + beta1(p_k(i)) * nNDVI(i) +
beta2(p_k(i)) * nNDVI(i) * nNDVI(i) + beta3(p_k(i)) * nGDD(i) + beta4(p_k(i))
* nGDD(i) * nGDD(i) + Omega2(k)) / (exp(alpha0 + alpha1(tp_k(i)) * nBias(i) +
alpha2(tp_k(i)) * nPopD(i)) + 1);
    }
    for(int k=0; k<nodes3; k++, g++){
      jnll_comp(5) += weight(g) * exp(alpha0 + alpha1(tp_k(g)) * nBias(g) +
alpha2(tp_k(g)) * nPopD(g) + beta(tp_k(g)) + beta1(p_k(g)) * nNDVI(g) +
beta2(p_k(g)) * nNDVI(g) * nNDVI(g) + beta3(p_k(g)) * nGDD(g) + beta4(p_k(g))
* nGDD(g) * nGDD(g) + Omega3(k)) / (exp(alpha0 + alpha1(tp_k(g)) * nBias(g) +
alpha2(tp_k(g)) * nPopD(g)) + 1);
    }
  }

  // Intensity function @ presence-only observations
  int h=0;
  for(int i=0; i<nobs1; i++, h++){
    jnll_comp(6) -= alpha0 + alpha1(tp_i(h)) * Bias(h) + alpha2(tp_i(h)) *
PopD(h) + beta(tp_i(h)) + beta1(p_i(h)) * NDVI(h) + beta2(p_i(h)) * NDVI(h) *
NDVI(h) + beta3(p_i(h)) * GDD(h) + beta4(p_i(h)) * GDD(h) * GDD(h) + omg1(i)
- log(exp(alpha0 + alpha1(tp_i(h)) * Bias(h) + alpha2(tp_i(h)) * PopD(h)) +
1);
  }

  for(int i=0; i<nobs2; i++, h++){
    jnll_comp(7) -= alpha0 + alpha1(tp_i(h)) * Bias(h) + alpha2(tp_i(h)) *
PopD(h) + beta(tp_i(h)) + beta1(p_i(h)) * NDVI(h) + beta2(p_i(h)) * NDVI(h) *
NDVI(h) + beta3(p_i(h)) * GDD(h) + beta4(p_i(h)) * GDD(h) * GDD(h) + omg2(i)
- log(exp(alpha0 + alpha1(tp_i(h)) * Bias(h) + alpha2(tp_i(h)) * PopD(h)) +
1);
  }

  for(int i=0; i<nobs3; i++, h++){
```

```
    jnll_comp(8) -= alpha0 + alpha1(tp_i(h)) * Bias(h) + alpha2(tp_i(h)) *
PopD(h) + beta(tp_i(h)) + beta1(p_i(h)) * NDVI(h) + beta2(p_i(h)) * NDVI(h) *
NDVI(h) + beta3(p_i(h)) * GDD(h) + beta4(p_i(h)) * GDD(h) * GDD(h) + omg3(i)
- log(exp(alpha0 + alpha1(tp_i(h)) * Bias(h) + alpha2(tp_i(h)) * PopD(h)) +
1);
  }

  // Intensity function @ structured sampling sites
  vector<Type> lambda(ncount);
  int f=0;
  for(int i=0; i<ncount1; i++, h++, f++){
    lambda(f) = area(f) * exp(beta(tp_i(h)) + beta1(p_i(h)) * NDVI(h) +
beta2(p_i(h)) * NDVI(h) * NDVI(h) + beta3(p_i(h)) * GDD(h) + beta4(p_i(h)) *
GDD(h) * GDD(h) + omg1(i+nobs1));
    jnll_comp(9) -= dpois(counts(f), lambda(f), true);
  }

  for(int i=0; i<ncount2; i++, h++, f++){
    lambda(f) = area(f) * exp(beta(tp_i(h)) + beta1(p_i(h)) * NDVI(h) +
beta2(p_i(h)) * NDVI(h) * NDVI(h) + beta3(p_i(h)) * GDD(h) + beta4(p_i(h)) *
GDD(h) * GDD(h) + omg2(i+nobs2));
    jnll_comp(10) -= dpois(counts(f), lambda(f), true);
  }

  for(int i=0; i<ncount3; i++, h++, f++){
    lambda(f) = area(f) * exp(beta(tp_i(h)) + beta1(p_i(h)) * NDVI(h) +
beta2(p_i(h)) * NDVI(h) * NDVI(h) + beta3(p_i(h)) * GDD(h) + beta4(p_i(h)) *
GDD(h) * GDD(h) + omg3(i+nobs3));
    jnll_comp(11) -= dpois(counts(f), lambda(f), true);
  }

  // Joint NLL
  Type jnll = jnll_comp.sum();

  // Reporting
  REPORT( tau );
  REPORT( kappa );
  REPORT( sigma );
  ADREPORT( tau );
  ADREPORT( kappa );
  ADREPORT( sigma );

  REPORT( jnll_comp );
  REPORT( jnll );

  return jnll;
}
```

Literature Cited

Center for International Earth Science Information Network - CIESIN - Columbia University (2018) Gridded Population of the World, Version 4 (GPWv4): Population Density, Revision 11. Palisades, NY: NASA Socioeconomic Data and Applications Center (SEDAC). DOI: 10.7927/H49C6VHW.

Didan, K. (2015) MOD13A2 MODIS/Terra Vegetation Indices 16-Day L3 Global 1km SIN Grid V006 [Data set]. NASA EOSDIS Land Processes DAAC. DOI: 10.5067/MODIS/MOD13A2.006.

Geldmann, J., Heilmann-Clausen, J., Holm, T.E., Levinsky, I., Markuseen, B., Olsen, K., … Tøttrup, A.P. (2016) What determines spatial bias in citizen science? Exploring four recording schemes with different proficiency requirements. Diversity and Distributions, 22: 1139-1149.

Krainski, E.T., Gómez-Rubio, V., Bakka, H., Lenzi, A., Castro-Camilo, D., Simpson, D., … Rue, H. (2018) Advanced spatial modeling with stochastic partial differential equations using R and INLA. Chapman and Hall, New York.

McMaster, G.S. and Wilhelm, W.W. (1997) Growing degree-days: one equation, two interpretations. Agricultural and Forest Meteorology, 87: 291-300.

Simpson, D., Illian, J.B., Lindgren, F., Sørbye, S.H., and Rue, H. (2016) Going off grid: computationally efficient inference for log-Gaussian Cox processes. Biometrika, 103: 49-70.

Thornton, M.M., R. Shrestha, Y. Wei, P.E. Thornton, S. Kao, and Wilson, B.E. (2020) Daymet: Daily Surface Weather Data on a 1-km Grid for North America, Version 4. ORNL DAAC, Oak Ridge, Tennessee, USA. DOI: 10.3334/ORNLDAAC/1840.