A GLMM approach for combining multiple relative abundance surfaces

Paul B. Conn, Jay M. Ver Hoef, Brett T. McClintock, Brian Brost, and Devin S. Johnson

Appendix S1: Combining relative abundance surfaces with different spatial support: theory and a vignette

In the main article, we focused on models where the spatial support of each surface was the same (i.e., each surface had the same pixel resolution and spatial extent). In this appendix, we demonstrate how our GLMM approach can be adapted to accommodate spatial surfaces where predictions were made at different spatial scales. This can include cases where pixel sizing is different, as well as cases where different areas are sampled (although we only consider the case where surfaces overlap to some degree).

In order to accommodate this change, we make the following modifications. First, denote the spatial domain of interest (i.e., the study area for inference) as $\mathcal{D}$, and denote the spatial domain of the ith surface by $\mathcal{D}_i$. We shall require that $\mathcal{D}_i \subseteq \mathcal{D} \ \forall i$. In practice, this just means that the combined map we're trying to estimate should include all areas covered in any of our spatial surfaces. Next, we discretize $\mathcal{D}$ to include S equal sized spatial cells, and assume that each of the relative abundance surfaces has been discretized into S_i spatial cells (these need not be of equal area). Finally, we reformulate eqn 1 from the main text as

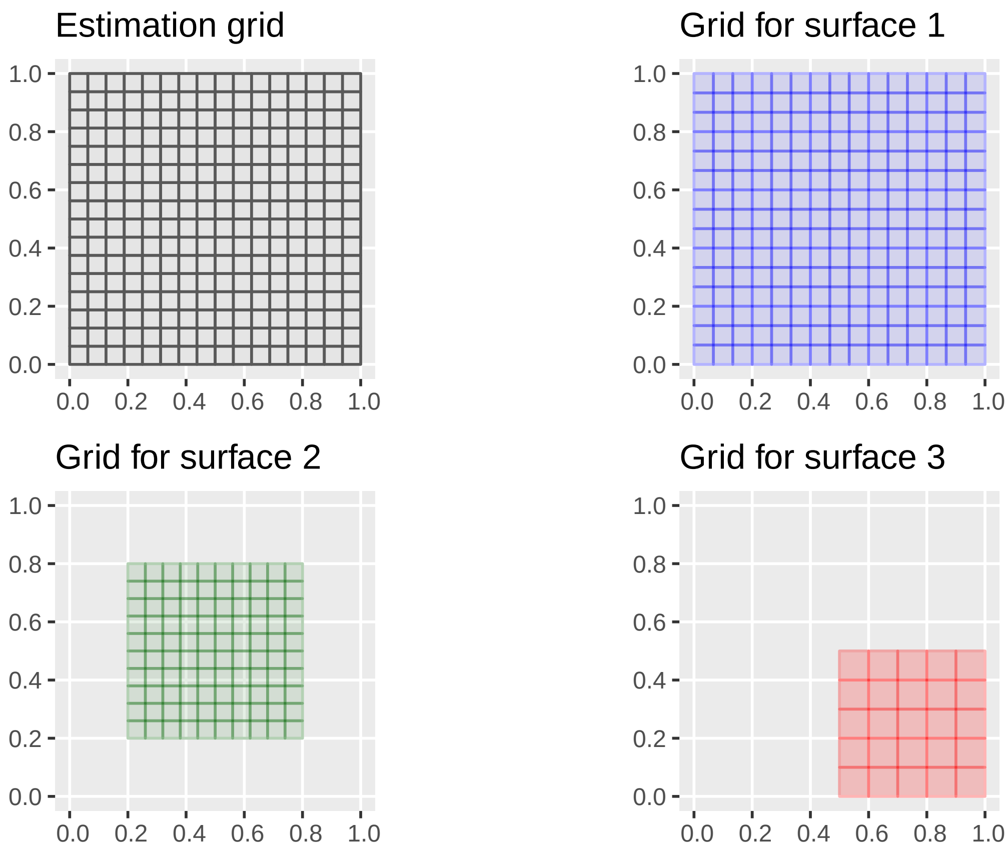
$$\mathbf{y}_i = \alpha_i + \log(\mathbf{A}_i \exp(\boldsymbol{\mu})) + \boldsymbol{\xi}_i + \boldsymbol{\epsilon}_i,$$

making several changes to definitions. First, the vectors $\mathbf{y}_i$, $\boldsymbol{\xi}_i$, and $\boldsymbol{\epsilon}_i$ now have length S_i (that is, the number of spatial cells is allowed to vary by surface). The elements of $\boldsymbol{\mu}$, μ_k, now refer to log-scale relative abundance in each of the S cells (at the scale we want to make inference). The $S_i \times S$ matrix $\mathbf{A}_i$ is used to link these "true" log scale relative abundance values to the (possibly mismatched) spatial cells associated with spatial surface i. In particular, if we let s_{ij} denote cell j from spatial surface i, and s_k denote pixel k of our desired relative abundance surface, we set the elements of $\mathbf{A}_i$, a_{jk}, equal to the area of cell s_{ij} that is made up of pixel s_k. This weighting is often termed "proportional allocation" or "areal weighting" in the context of geographic analysis (Gotway and Young 2002).

We shall now demonstrate estimation with this formulation on a simple example. We use a 16 x 16 underlying grid to generate data, which we will also use for the spatial support of $\boldsymbol{\mu}$. We will generate data for 3 different relative abundance surfaces that vary by grid spacing and spatial coverage of the survey area. The grid spacing of these surfaces is *not* nested; that is, there is fractional overlap between relative abundance grid cells and the $\boldsymbol{\mu}$ grid.

```
library(sf)
D = st_sfc(st_polygon(list(rbind(c(0, 0), c(1, 0), c(1, 1), c(0,
    0)))))
Grid_mu = st_make_grid(D, n = c(16, 16))  #Mu grid is 16x16 on unit square
D1 = D
Grid_i = vector("list", 3)
Grid_i[[1]] = st_make_grid(D1, n = c(15, 15))  #surface 1 is 15 x 15 on unit square
D2 = st_sfc(st_polygon(list(rbind(c(0.2, 0.2), c(0.8, 0.2), c(0.8,
    0.8), c(0.2, 0.2)))))
Grid_i[[2]] = st_make_grid(D2, n = c(10, 10))  #surface 2 is 10 x 10 inner square
D3 = st_sfc(st_polygon(list(rbind(c(0.5, 0), c(1, 0), c(1, 0.5),
    c(0.5, 0)))))
```

```
Grid_i[[3]] = st_make_grid(D3, n = c(5, 5))  #surface 3 is 5x5 on lower right quadrant


library(ggplot2)
Grid_mu_plot = ggplot() + geom_sf(data = Grid_mu) + ggtitle("Estimation grid")
Grid1_plot = ggplot() + geom_sf(data = Grid_i[[1]], color = alpha("blue",
    0.3), fill = alpha("blue", 0.1)) + xlim(0, 1) + ylim(0, 1) +
    ggtitle("Grid for surface 1")
Grid2_plot = ggplot() + geom_sf(data = Grid_i[[2]], colour = alpha("darkgreen",
    0.3), fill = alpha("darkgreen", 0.1)) + xlim(0, 1) + ylim(0,
    1) + ggtitle("Grid for surface 2")
Grid3_plot = ggplot() + geom_sf(data = Grid_i[[3]], color = alpha("red",
    0.3), fill = alpha("red", 0.2)) + xlim(0, 1) + ylim(0, 1) +
    ggtitle("Grid for surface 3")
gridExtra::grid.arrange(Grid_mu_plot, Grid1_plot, Grid2_plot,
    Grid3_plot, nrow = 2)
```



Our next step will be to construct $\mathbf{A}_i$ for each of the three relative abundance surfaces. We'll use some of the functionality of the “sf" package in R to calculate the overlap between each relative abundance grid and the various cells of the estimation grid.

```
A = Intersects = vector("list", 3)
cell_size = st_area(Grid_mu[1])
for (isurf in 1:3) {
    Intersects[[isurf]] = A[[isurf]] = 1 * st_intersects(Grid_i[[isurf]],
        Grid_mu, sparse = FALSE)
    for (icell in 1:nrow(A[[isurf]])) {
        Which_intersects = which(Intersects[[isurf]][icell, ] >
            0)
        for (itarget in 1:length(Which_intersects)) {
            A[[isurf]][icell, Which_intersects[itarget]] = st_area(st_intersection(Grid_i[[isurf]][icel
                Grid_mu[Which_intersects[itarget]]))/cell_size
        }
    }
}
```

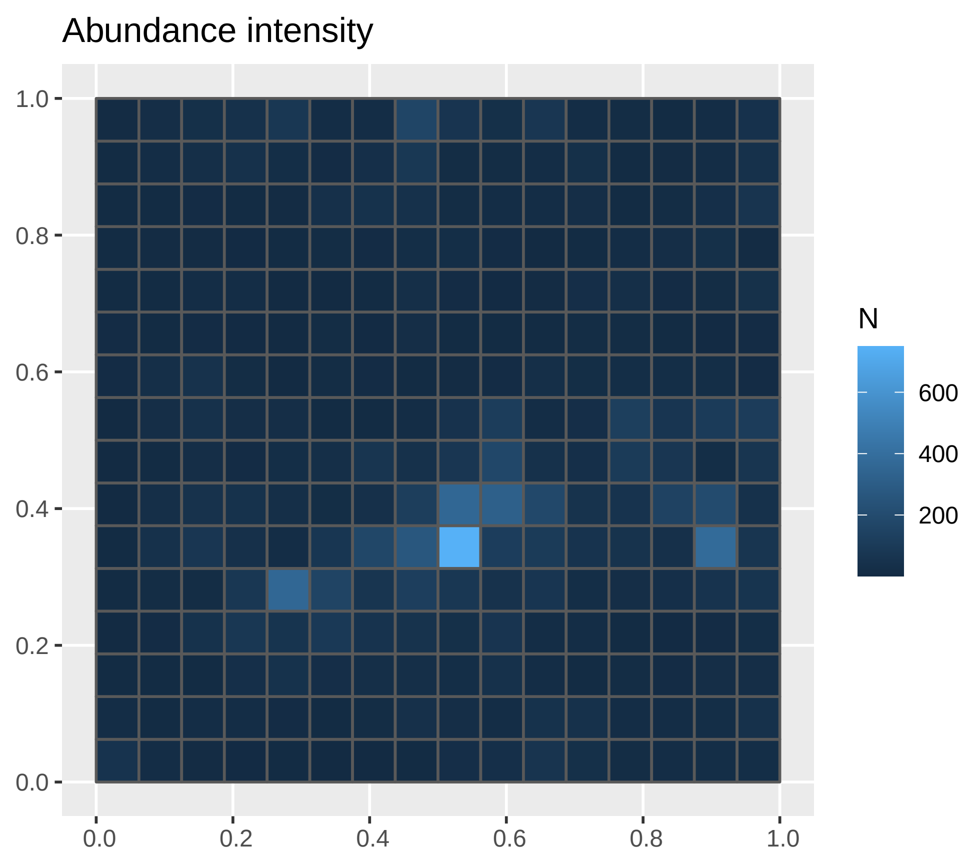
For instance, the sum of the first row of A_3 here is 2.56, which makes sense because each cell of $\mathcal{D}_3$ is 2.56 times as large as the cells associated with $\mathcal{D}$. In essence, the A_i matrices are 'picking' the elements of $\boldsymbol{\mu}$ that intersect with a particular relative abundance grid cell, and using the relative area of those intersecting cells to produce an expectation for different sized grid cells.

Now that we've got those set up, let's simulate true log-scale relative abundance values, $\boldsymbol{\mu}$:

```
set.seed(2022)  #setting a random # seed so we can duplicate results
n_s = length(Grid_mu)
XY_mu = st_coordinates(st_centroid(Grid_mu))
Dists = as.matrix(dist(XY_mu, diag = T, upper = T))
Cov_exp = fields::Exp.cov(XY_mu, XY_mu, aRange = 0.1)
L = chol(Cov_exp)
Mu = t(L) %*% rnorm(n_s, 0, 1)
Nu = exp(Mu)
Pi = Nu/(sum(Nu))
N = 10000  #true abundance
Exp_N = N * Pi  #expected abundance in Mu grid
```

Here's what expected abundance looks like for this example:

```
Grid_mu = st_sf(Grid_mu)
Grid_mu$N = Exp_N
ggplot() + geom_sf(data = Grid_mu, aes(fill = N)) + ggtitle("Abundance intensity")
```

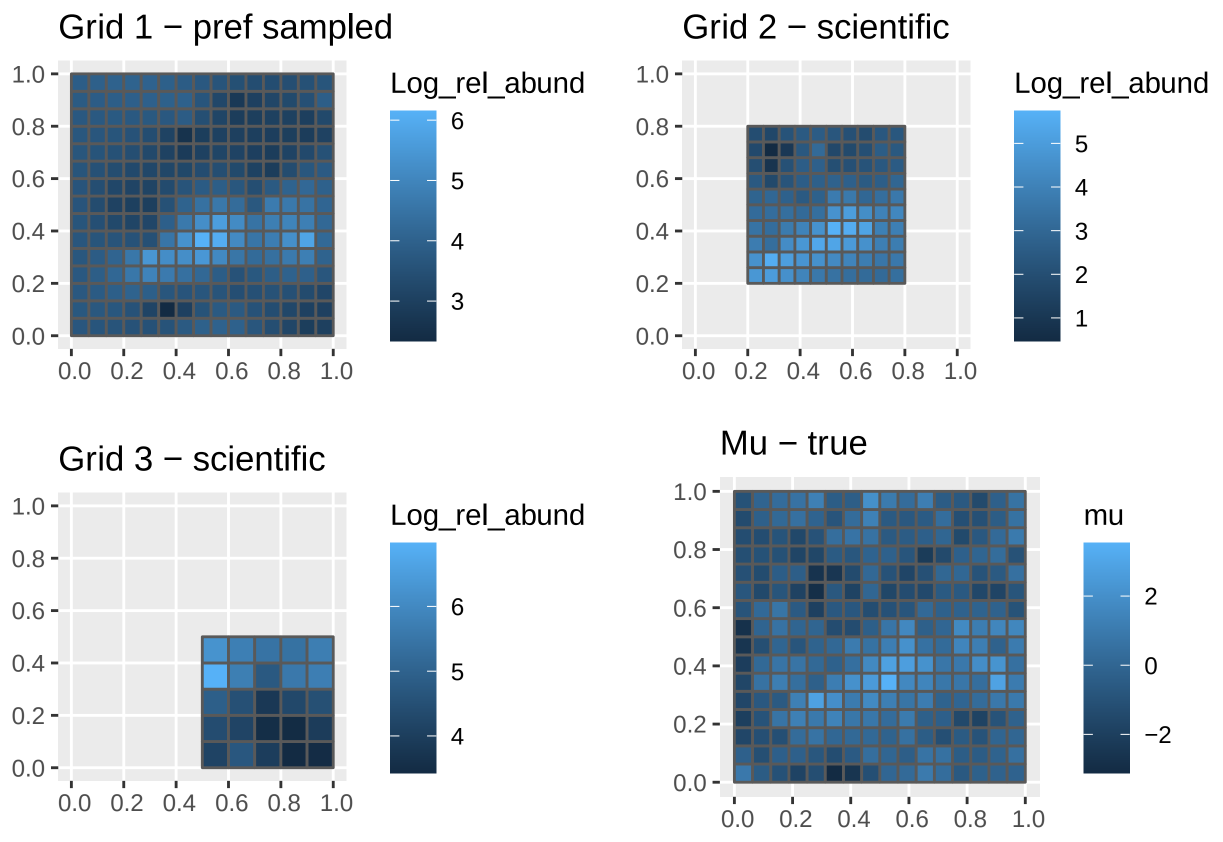


We will not typically be able to observe relative abundance (or density, or true abundance) directly, but must sample it in some fashion and attempt to estimate/predict what abundance is like in unsampled locations. We have already indicated the spatial extent and pixel size for three relative abundance surfaces that we will have to work with. We shall simulate some counts over these domains, assuming some level of random sampling. However, we will assume that the first (and largest) relative abundance surface is developed from citizen science data and is preferentially sampled - that is, observers are more likely to collect data where they expect to actually see the organism of interest (Conn, Thorson, and Johnson 2017). We will then assume the 2nd and 3rd surfaces are developed from scientific surveys where random sampling is employed and "collect" some counts for each of these study areas and estimate relative abundance surfaces for each one, using models with spatial autocorrelation to account for changing abundance over space. We will do this by conducting virtual count surveys in different cells, assuming that abundance intensity (i.e., exp($\boldsymbol{\mu}$)), or its analogue in larger grid cells) is constant within the cells surveyed. We will also fit Poisson models to each surface's count data that include spatially autocorrelated random effects. Our goal in this step is to estimate a spatial surface for each of the three sets of count data, as well as a variance-covariance matrix for log-scale predictions.

```
library(TMB)
library(INLA)
TmbFile1 = "../src/fit_1_surface"
compile(paste0(TmbFile1, ".cpp"), "-O1 -g", DLLFLAGS = "")
dyn.load(dynlib(TmbFile1))

XY = Exp_N_i = Sampled = Counts = Pred_surf = VC_surf = Mesh_xi = M0 = M1 = M2 = vector("list",
    3)
N_sampled = c(50, 50, 20)  #number of cells sampled in each grid
N_cells = rep(0, 3)
for (igrid in 1:3) {
    XY[[igrid]] = st_coordinates(st_centroid(Grid_i[[igrid]]))
    Exp_N_i[[igrid]] = A[[igrid]] %*% Exp_N  #expected abundance in each relative abundance grid
    N_cells[igrid] = nrow(XY[[igrid]])
    if (igrid == 1)
        Sampled[[igrid]] = sample(c(1:N_cells[igrid]), N_sampled[1],
            prob = Exp_N_i[[1]])  #preferential sampling for surface 1
 else Sampled[[igrid]] = sample(c(1:N_cells[igrid]), N_sampled[igrid])
    Counts[[igrid]] = rpois(N_sampled[igrid], Exp_N_i[[igrid]][Sampled[[igrid]]])
    # fit spatial model to each set of counts and predict
    # expected count in each grid cell
    Mesh_xi[[igrid]] = inla.mesh.create(as(st_centroid(Grid_i[[igrid]]),
        "Spatial"))
    spde <- (inla.spde2.matern(Mesh_xi[[igrid]], alpha = 2)$param.inla)[c("M0",
        "M1", "M2")]
    M0[[igrid]] = spde$M0
    M1[[igrid]] = spde$M1
    M2[[igrid]] = spde$M2
    n_knots = nrow(spde$M0)
    Data1 <- list(C_i = Counts[[igrid]], S_i = Sampled[[igrid]] -
        1, M0 = spde$M0, M1 = spde$M1, M2 = spde$M2, A_s = rep(1,
        N_cells[igrid]), P_i = rep(1, N_sampled[igrid]), X_s = matrix(1,
        N_cells[igrid], 1), flag = 1, n_i = N_sampled[igrid],
        n_s = N_cells[igrid], n_eta = n_knots, Eta_index = c(1:N_cells[igrid]) -
            1, options = c(0, 0), beta_pri = c(0, 1), matern_pri = c(0,
            0, 0, 0))
    Params1 <- list(Beta = 0, log_tau = 0, log_kappa = 0, Etainput_s = rep(0,
        n_knots))
    Random1 <- "Etainput_s"
    Map1 <- NULL
    Obj = MakeADFun(data = Data1, parameters = Params1, random = Random1,
        map = Map1, DLL = "fit_1_surface", silent = FALSE)
    Obj$fn(Obj$par)
    Lower = -50  #trying to prevent -Inf,Inf bounds resulting in nlminb failure (NaN gradient)
    Upper = 50
    Opt = nlminb(start = Obj$par, objective = Obj$fn, gradient = Obj$gr,
        lower = Lower, upper = Upper, control = list(trace = 1,
            eval.max = 500, iter.max = 500))  #
    Report = Obj$report()

    SD = sdreport(Obj, bias.correct = FALSE)
    Pred_surf[[igrid]] = SD$value
    VC_surf[[igrid]] = SD$cov
    diag(VC_surf[[igrid]]) = diag(VC_surf[[igrid]]) + 1e-07  #Tikhonov regularization so invertible
}
```

Now we have log-scale relative abundance predictions and associated variance-covariance matrices for each grid. We sill now plot them beefore we use them within our estimation architecture. We will also plot “true” log-scale relative abundance so we can have something to compare them to. Note the difference in scale when comparing plots - this arises because larger cells have a greater expected abundance than smaller ones.

```
for (igrid in 1:3) {
    Grid_i[[igrid]] = st_sf(Grid_i[[igrid]])
    Grid_i[[igrid]]$Log_rel_abund = Pred_surf[[igrid]]
}
plot1 = ggplot() + geom_sf(data = Grid_i[[1]], aes(fill = Log_rel_abund)) +
    ggtitle("Grid 1 - pref sampled") + xlim(0, 1) + ylim(0, 1)
plot2 = ggplot() + geom_sf(data = Grid_i[[2]], aes(fill = Log_rel_abund)) +
    ggtitle("Grid 2 - scientific") + xlim(0, 1) + ylim(0, 1)
plot3 = ggplot() + geom_sf(data = Grid_i[[3]], aes(fill = Log_rel_abund)) +
    ggtitle("Grid 3 - scientific") + xlim(0, 1) + ylim(0, 1)
Grid_mu$mu = Mu
plot4 = ggplot() + geom_sf(data = Grid_mu, aes(fill = mu)) +
    xlim(0, 1) + ylim(0, 1) + ggtitle("Mu - true")
gridExtra::grid.arrange(plot1, plot2, plot3, plot4, nrow = 2)
```



Now we will run these through our estimation machinery. We will enable process errors ($\boldsymbol{\xi}$) for surface 1 (this was the surface generated from preferentially sampled count data), but will set them to zero for surfaces 2 and 3. We will need to compile TMB code with our data, as well as options for which process error parameters to estimate. We will also need to use INLA to set up our GMRF bases to enable spatial covariance estimation given the various grid sizes (this is only needed for $\boldsymbol{\mu}$ and $\boldsymbol{\xi}_1$ in this example because $\boldsymbol{\xi}_2$ and $\boldsymbol{\xi}_3$ are set to zero).

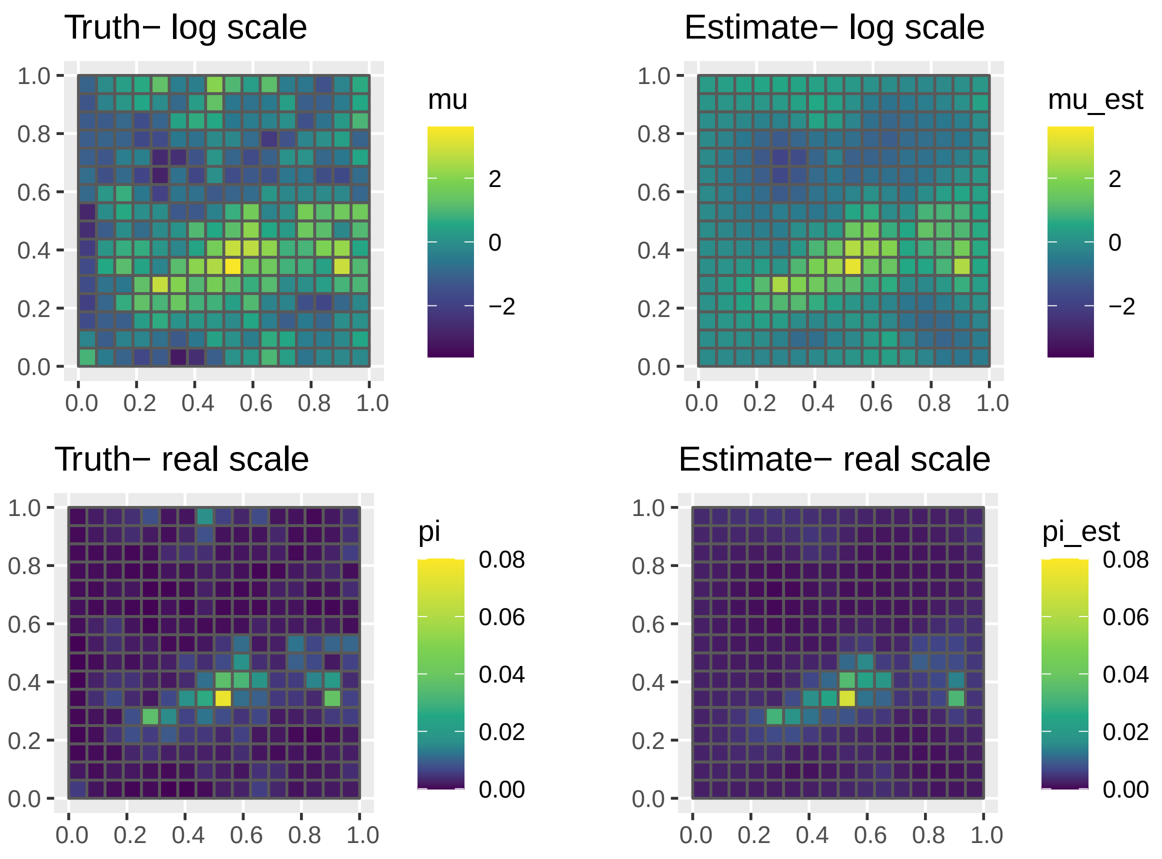
```
TmbFile = "../src/fit_multiple_surfaces_misalign"
compile(paste0(TmbFile, ".cpp"), "-O1 -g", DLLFLAGS = "")
dyn.load(dynlib(TmbFile))

mesh_mu = inla.mesh.create(XY_mu)
spde_mu <- (inla.spde2.matern(mesh_mu, alpha = 2)$param.inla)[c("M0",
    "M1", "M2")]

mesh_xi = inla.mesh.create(XY[[1]])
```

```r
spde_xi <- (inla.spde2.matern(mesh_xi, alpha = 2)$param.inla)[c("M0",
    "M1", "M2")]

n_surf = 3
n_s_mu = n_s
n_s = c(length(Pred_surf[[1]]), length(Pred_surf[[2]]), length(Pred_surf[[3]]))
n_s_max = max(n_s)
Y_i = matrix(0, n_s_max, n_surf)
Omega_Y = array(0, dim = c(n_s_max, n_s_max, n_surf))
A_i = array(0, dim = c(n_s_max, n_s_mu, n_surf))
for (isurf in 1:n_surf) {
    Y_i[1:n_s[isurf], isurf] = Pred_surf[[isurf]]
    Omega_Y[1:n_s[isurf], 1:n_s[isurf], isurf] = solve(VC_surf[[isurf]])
    A_i[1:n_s[isurf], , isurf] = A[[isurf]]
}

n_knots_xi = c(nrow(spde_xi$M0), 0, 0)
n_knots_mu = nrow(spde_mu$M0)


Data <- list(Y_i = Y_i, Omega_Y = Omega_Y, M0_mu = spde_mu$M0,
    M1_mu = spde_mu$M1, M2_mu = spde_mu$M2, M0 = spde_xi$M0,
    M1 = spde_xi$M1, M2 = spde_xi$M2, A_i = A_i, flag = 1, n_s_mu = n_s_mu,
    n_s = n_s, n_surf = n_surf, n_knots_xi = n_knots_xi, options = c(0,
        0))
Params <- list(log_alpha = rep(0, n_surf), log_tau_mu = 0, log_kappa_mu = 0,
    log_tau_xi = rep(0, n_surf), log_kappa_xi = rep(0, n_surf),
    Mu_s = rep(0, n_knots_mu), Xi_s = matrix(0, nrow(spde_xi$M0),
        n_surf))
Random <- c("Mu_s", "Xi_s")

# controlling which parameters to fix (NAs aren't
# estimated)
Map_xi = matrix(NA, nrow(spde_xi$M0), n_surf)
Map_xi[1:n_s[1], 1] = c(1:n_s[1])
Map_log_kappa_xi = rep(NA, n_surf)
Map_log_kappa_xi[1] = n_s[1] + 1
Map_log_tau_xi = rep(NA, n_surf)
Map_log_tau_xi[1] = n_s[1] + 2
Map = list(Xi_s = factor(Map_xi), log_kappa_xi = factor(Map_log_kappa_xi),
    log_tau_xi = factor(Map_log_tau_xi))


Obj = MakeADFun(data = Data, parameters = Params, random = Random,
    map = Map, DLL = "fit_multiple_surfaces_misalign", silent = FALSE)
Obj$fn(Obj$par)
Lower = -50  #trying to prevent -Inf,Inf bounds resulting in nlminb failure (NaN gradient)
Upper = 50
Opt = nlminb(start = Obj$par, objective = Obj$fn, gradient = Obj$gr,
    lower = Lower, upper = Upper, control = list(trace = 1, eval.max = 500,
        iter.max = 500))  #
Report = Obj$report()
```

Now, we will visualize our estimated "combined" relative abundance index on the the grid specified for $\boldsymbol{\mu}$.

We will also plot truth next to the estimated surface on both the log scale and real scale.

```r
Grid_mu$mu_est = Report$Mu_s[1:n_s_mu]
Grid_mu$pi_est = Report$Pi
Grid_mu$pi = Grid_mu$N/sum(Grid_mu$N)
plot1 = ggplot() + geom_sf(data = Grid_mu, aes(fill = mu)) +
    ggtitle("Truth- log scale") + scale_fill_viridis_c(limits = c(-3.6,
    3.6), breaks = c(-4, -2, 0, 2, 4), values = c(0, 0.3, 0.5,
    0.7, 1))
plot2 = ggplot() + geom_sf(data = Grid_mu, aes(fill = mu_est)) +
    ggtitle("Estimate- log scale") + scale_fill_viridis_c(limits = c(-3.6,
    3.6), breaks = c(-4, -2, 0, 2, 4), values = c(0, 0.3, 0.5,
    0.7, 1))
plot3 = ggplot() + geom_sf(data = Grid_mu, aes(fill = pi)) +
    ggtitle("Truth- real scale") + scale_fill_viridis_c(limits = c(0,
    0.08), breaks = c(0, 0.02, 0.04, 0.06, 0.08), values = c(0,
    0.1, 0.2, 0.5, 1))
plot4 = ggplot() + geom_sf(data = Grid_mu, aes(fill = pi_est)) +
    ggtitle("Estimate- real scale") + scale_fill_viridis_c(limits = c(0,
    0.08), breaks = c(0, 0.02, 0.04, 0.06, 0.08), values = c(0,
    0.1, 0.2, 0.5, 1))
gridExtra::grid.arrange(plot1, plot2, plot3, plot4, nrow = 2)
```



References

Conn, Paul B, James T Thorson, and Devin S Johnson. 2017. “Confronting Preferential Sampling When

Analysing Population Distributions: Diagnosis and Model-Based Triage.” *Methods in Ecology and Evolution* 8 (11): 1535–46.

Gotway, Carol A, and Linda J Young. 2002. “Combining Incompatible Spatial Data.” *Journal of the American Statistical Association* 97 (458): 632–48.