Species-level Spatial Model Simulations

Russell Dinnage

2 July 2019

Species-level Spatial Model

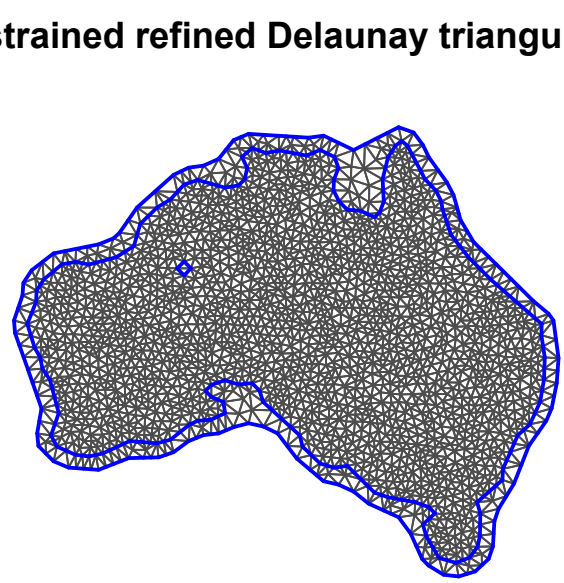
One of the unique things about the spatiophylogenetic model we propose in the manuscript is that it can fit a spatial model when the response is only measured at the species-level, but where each species has multiple spatial occurrence measurements. As we noted in the manuscript, without this unique spatial component, our model collapses to a relatively standard phylogenetic model. Here, we will do some simulations to get an idea of how this unique spatial component of our model functions. First we will simulate some data from the Matern covariance function, based on the mesh that we used in the manuscript. So we will simulate our data over Australia. This is mainly for convenience, since we already have designed a mesh, and is otherwise not meaningful. We base these simulation off the examples found at https://haakonbakka.bitbucket.io/btopic122.html, written by Haakon Bakka. Let's load our necessary libraries:

```
library(INLA)
library(fields)
library(ggplot2)
library(scico)
library(readr)
library(rnaturalearth)
library(sf)
library(dplyr)
library(sp)
library(rmapshaper)
library(ids)
library(patchwork)
set.seed(201803)
inla.seed = sample.int(n=1E6, size=1)
```

Next we load our *Hakea* occurrence points which we use to reconstruct the mesh we used in the manuscript. We'll also grab a map of Australia from `rnaturalearth`

```
hakea_occ <- read_csv("data/Hakea_occ_clean.csv")
aus_coast <- ne_countries(country = "Australia", scale = 10, returnclass = "sf") %>%
  rmapshaper::ms_filter_islands(1e+10)

## make mesh
coordsy <- SpatialPoints(hakea_occ[ , c("longitude", "latitude")],
                         proj4string = CRS("+proj=longlat +ellps=WGS84 +datum=WGS84"))

boundary.loc <- coordsy
boundary <- list(
  inla.nonconvex.hull(coordinates(boundary.loc), 0.5, crs=inla.CRS("longlat")),
```

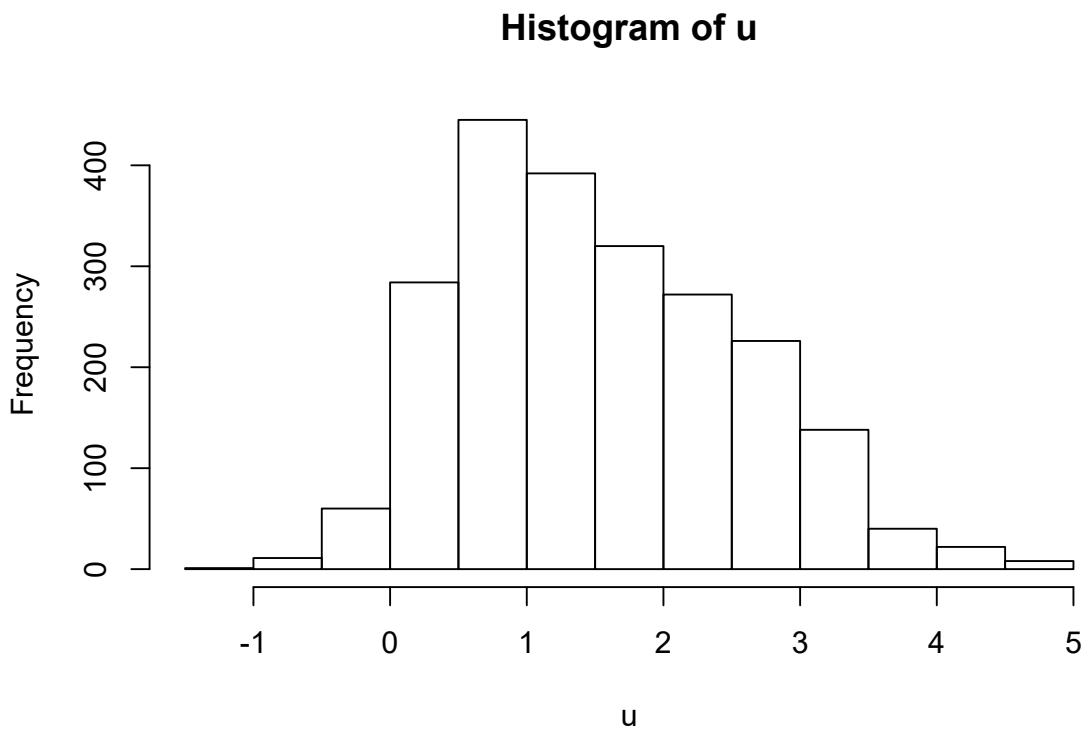
```
  inla.nonconvex.hull(coordinates(boundary.loc), 1.6, crs=inla.CRS("longlat") ))

## Build the mesh:
spatial_mesh <- inla.mesh.2d(boundary=boundary,
                             max.edge=c(1, 2),
                             min.angle=26,
                             cutoff=0.5, ## Filter away adjacent points.
                             offset=c(0.5, 1.6)
)
plot(spatial_mesh, asp = 1)
```



Now we can simulate some data from the Matern covariance on the mesh. We will set the range and standard deviation parameter of the Matern covariance to an arbitrary value, and see what it looks like. Later we will try different values.

```
range <- 15
sigma.u <- 1

## This function requires specifying a prior, although they are not used for this application
spde = inla.spde2.pcmatern(spatial_mesh, prior.range = c(.5, .5), prior.sigma = c(.5, .5))

Qu = inla.spde.precision(spde, theta=c(log(range), log(sigma.u)))
u = inla.qsample(n = 1, Q = Qu, seed = inla.seed)

if(nrow(u) > 1) {
    u <- u[ , 1]
}
```

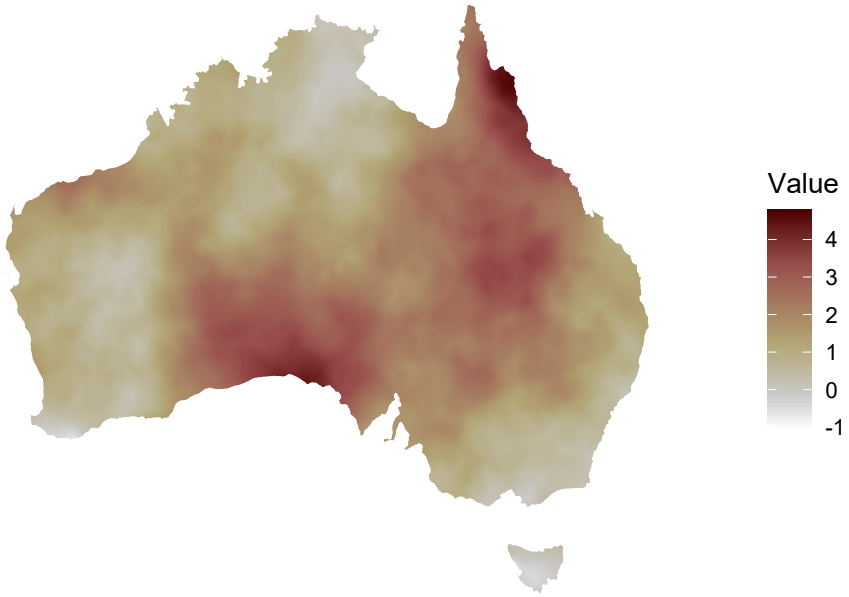
```r
hist(u)
```



Okay, now let's setup a function to let us quickly visualize a simulated spatial field like the one we just made.

```r
plot_spatial_field = function(u, spatial_mesh, aus_coast){
  stopifnot(length(u) == spatial_mesh$n)
  proj = inla.mesh.projector(spatial_mesh, dims=c(300, 300))
  u_proj = inla.mesh.project(proj, u)

  u_df <- dplyr::tibble(x = proj$lattice$loc[ , 1],
                        y = proj$lattice$loc[ , 2],
                        u = as.vector(u_proj))

  ##create a "sea mask" to cover values extrapolated into the ocean
  sea_mask <- as(raster::extent(c(min(u_df$x),
                                  max(u_df$x),
                                  min(u_df$y),
                                  max(u_df$y))), "SpatialPolygons")
  sea_mask <- rgeos::gBuffer(sea_mask, width = 1)
  suppressWarnings(au_sea_shp <- rgeos::gDifference(sea_mask, sf::as_Spatial(aus_coast)) %>%
    sf::st_as_sf() %>%
      sf::st_set_crs(4326))

  ggplot(u_df, aes(x, y)) +
    geom_raster(aes(fill = u)) +
    geom_sf(data = au_sea_shp, fill = "white", colour = "white", inherit.aes = FALSE) +
    scale_fill_scico(name = "Value") +
```

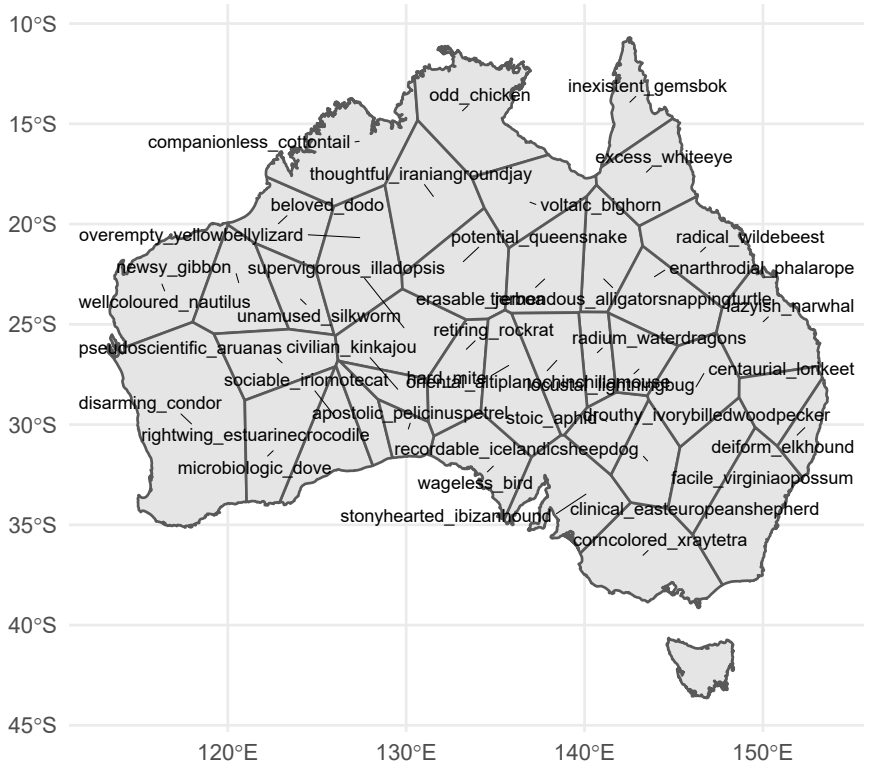
```
    xlab("") +
    ylab("") +
    theme_minimal() +
    theme(axis.text = element_blank(),
          panel.grid = element_blank())
}

## Try it out!

plot_spatial_field(u, spatial_mesh, aus_coast)
```



Okay, looks good! Now we need to simulate some species occurrence points and calculate the species-level value for this spatial field. There is a million way we could simulate species ranges, but we have to decide on one. We are interested in testing the effects of within species sample sizes, and the degree of overlap of species ranges on the estimated spatial field. So a sensible place to start is to generate species ranges with no overlap. Then we can expand these ranges or shift them around to generate more overlap. A simple way to do this is to throw down some points on our map and then use a voronoi tesselation to generate non-overlapping ranges. Let's do that.

```
## how many species?
n_spec <- 40

range_sample <- aus_coast %>%
  sf::st_sample(n_spec) %>%
  sf::st_combine() %>%
  sf::st_voronoi(envelope = aus_coast$geometry[[1]]) %>%
  sf::st_cast() %>%
```

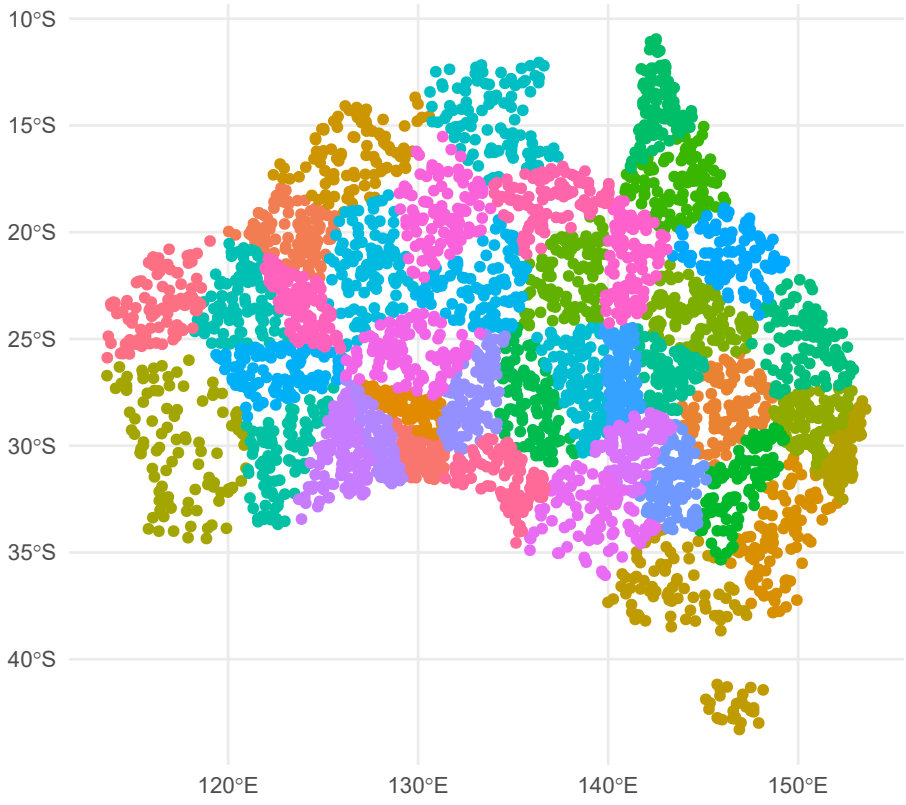
```
  data.frame(geometry = .) %>%
  sf::st_sf(.) %>%
  sf::st_transform(4326) %>%
  sf::st_intersection(., aus_coast %>% dplyr::select(geometry)) %>%
  sf::st_cast() %>%
  dplyr::mutate(species = ids::adjective_animal(n = n()))

ggplot(range_sample) +
  geom_sf() +
  ggrepel::geom_text_repel(
      aes(label = species, geometry = geometry),
      data = range_sample,
      stat = "sf_coordinates",
      min.segment.length = 0,
      segment.size = 0.1,
      size = 2.5,
      box.padding = 0.25,
      force = 2,
      max.iter = 5000
    ) +
  xlab("") +
  ylab("") +
  theme_minimal()
```



Okay, species ranges! Now to sample points from each range, put them into a `tibble`, and then sample their values on our simulated spatial field. Lastly, we will aggregate the data to a single value for each species. For this we will use the mean values for each species, which is the way what we assume in our spatiophylogenetic

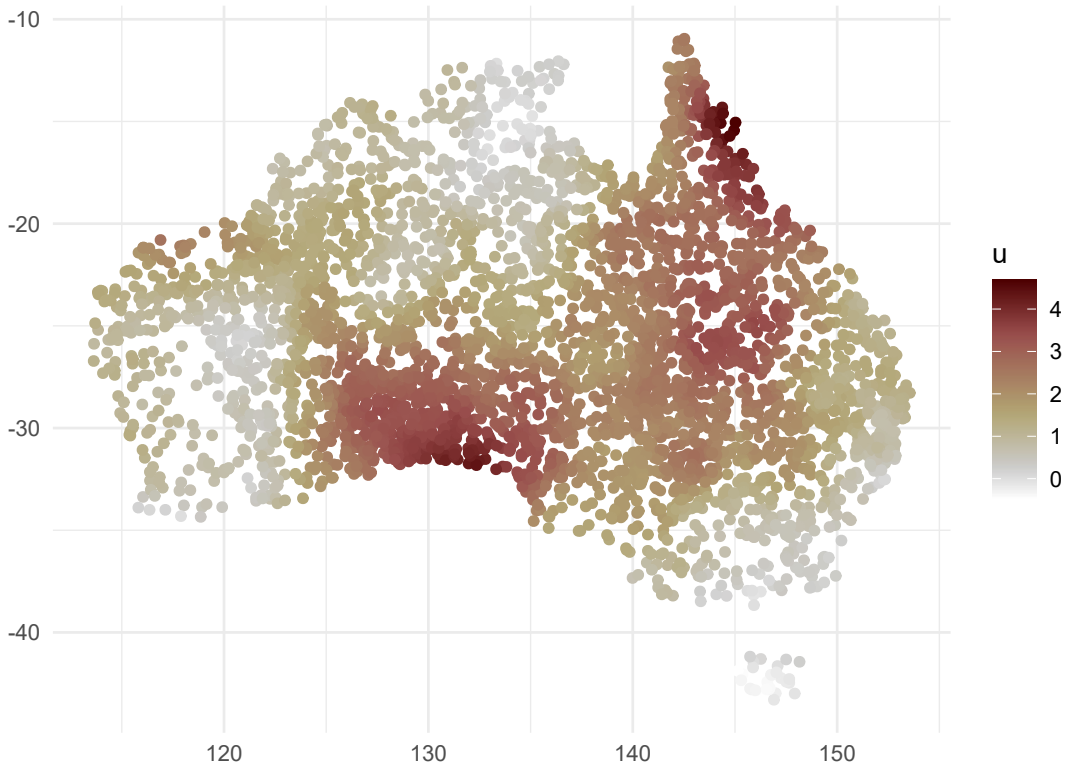
model plus some species-level noise thrown in.

```
## number of points per species
n_pnt_per_spec <- 100

## species-level noise
noise_sd <- 0.1

size <- rep(n_pnt_per_spec, n_spec)
suppressMessages(point_sample <- range_sample %>%
  sf::st_sample(size = size) %>%
  sf::st_sf(ID = rep(seq_along(size), size),
            geometry = .) %>%
  dplyr::mutate(species = range_sample$species[ID]))

ggplot(point_sample) +
  geom_sf(aes(colour = species)) +
  theme_minimal() +
  theme(legend.position = "none")
```




```
pnt_df <- point_sample %>%
  sf::st_coordinates() %>%
  dplyr::as_tibble() %>%
  dplyr::mutate(species = point_sample$species)

## project to mesh
proj <- inla.mesh.projector(spatial_mesh, loc = as.matrix(pnt_df %>% dplyr::select(X, Y)))
```

```
u_proj <- inla.mesh.project(proj, u)

pnt_df <- pnt_df %>%
  dplyr::mutate(u = u_proj)

ggplot(pnt_df, aes(X, Y)) +
  geom_point(aes(colour = u)) +
  scale_colour_scico() +
  coord_equal() +
  xlab("") +
  ylab("") +
  theme_minimal()
```



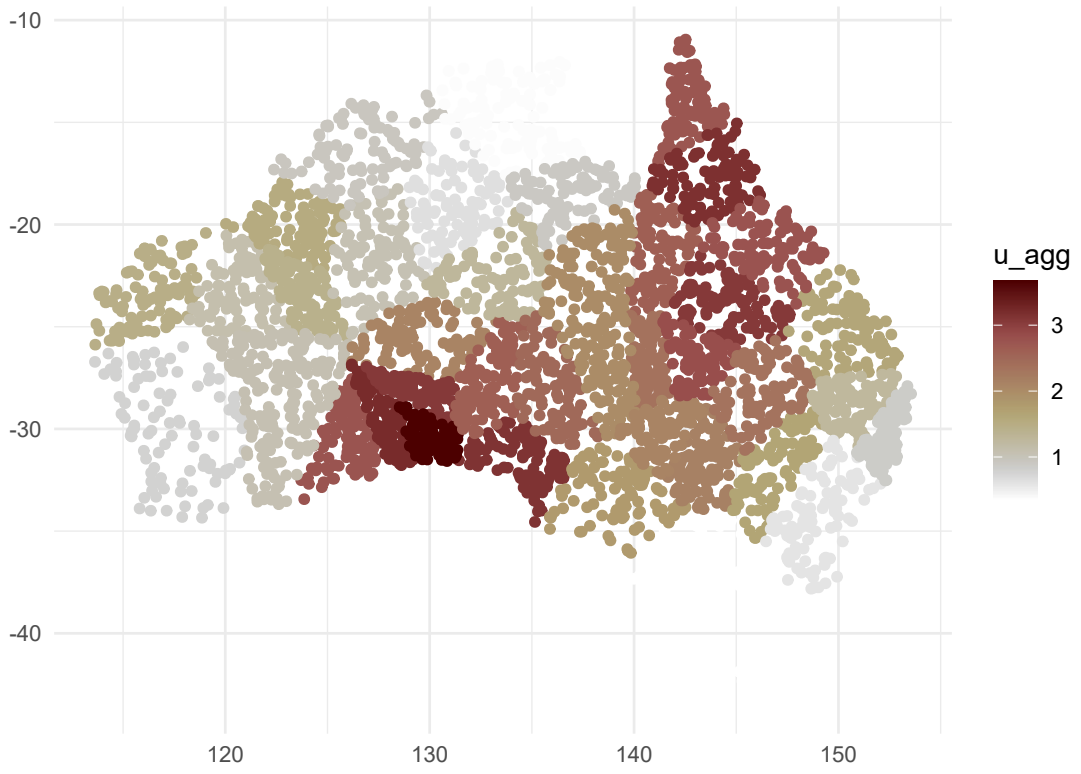

```
spec_df <- pnt_df %>%
  dplyr::group_by(species) %>%
  dplyr::summarise(u = mean(u)) %>%
  dplyr::mutate(u = u + rnorm(n(), sd = noise_sd)) ## add species-level noise

spec_df
```

```
## # A tibble: 40 x 2
##    species                        u
##    <chr>                      <dbl>
##  1 apostolic_pelicinuspetrel  3.67
##  2 beloved_dodo               1.54
##  3 centaurial_lorikeet        2.34
```

```
##  4 civilian_kinkajou             3.06
##  5 clinical_easteuropeanshepherd 0.611
##  6 companionless_cottontail      0.941
##  7 corncolored_xraytetra         0.362
##  8 deiform_elkhound              0.866
##  9 disarming_condor              0.787
## 10 drouthy_ivorybilledwoodpecker 1.26
## # ... with 30 more rows
```

So, once now that we have aggregated by species, what does our spatial data look like?

```
ggplot(pnt_df %>%
         dplyr::left_join(spec_df %>%
                            dplyr::rename(u_agg = u)),
       aes(X, Y)) +
  geom_point(aes(colour = u_agg)) +
  scale_colour_scico() +
  coord_equal() +
  xlab("") +
  ylab("") +
  theme_minimal()
```



Okay, now we have to setup our `INLA` model as we did in the manuscript (and also in the Supplementary tutorial, where you can get more details on this). For the purposes of testing the spatial effect we will use Gaussian errors for simulating and fitting the model. In the manuscript our response was Bernoulli distributed, which is generally a more difficult type of data to fit, because of much less information in a discrete variable with only two values. So, these simulation represent a best case scenario, to show that the models can capture what we are looking for them to capture.

```
## use the same priors we used in the manuscript
spde_fit <- inla.spde2.pcmatern(
      mesh = spatial_mesh, alpha = 2, ### mesh and smoothness parameter
      prior.range = c(2, 0.1), ### P(practic.range<2)=0.1
      prior.sigma = c(1, 0.1),
      constr = TRUE)

coords <- as.matrix(pnt_df[ , c("X", "Y")])
A <- inla.spde.make.A(spatial_mesh, loc = coords) ## this creates a sparse matrix
dim(A)
```

```
## [1] 4000 2219
```

```
specs <- spec_df$species

spec_A_rows <- lapply(specs, function(x) { ## apply to each species one at a time

  A[which(pnt_df$species == x), ] %>% ## get rows associated with the species
    apply(2, mean) ## average each column to return a single row with the same number of columns

}) %>%
  do.call(rbind, .) ## rbind the species back together into one matrix

## convert back to sparse matrix
spec_A_rows <- Matrix(spec_A_rows, sparse = TRUE)

## check some stuff
dim(spec_A_rows)
```

```
## [1]   40 2219
```

```
apply(spec_A_rows, 1, sum) %>% head
```

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

```
stack <- inla.stack(data = list(resp = spec_df$u),
                    A = list(spec_A_rows, 1), ## one A for each effect, 1 is shorthand, see below
                       effects=list(spat_id = 1:spde_fit$n.spde,
                                    intercept = rep(1, nrow(spec_df))),
                      tag='est')

stck.loc <- inla.stack(data = list(resp = NA),
                          A = list(spec_A_rows),
                          effects = list(spat_id = 1:spde_fit$n.spde), tag='loc')

big_stack <- inla.stack(stack, stck.loc)
```

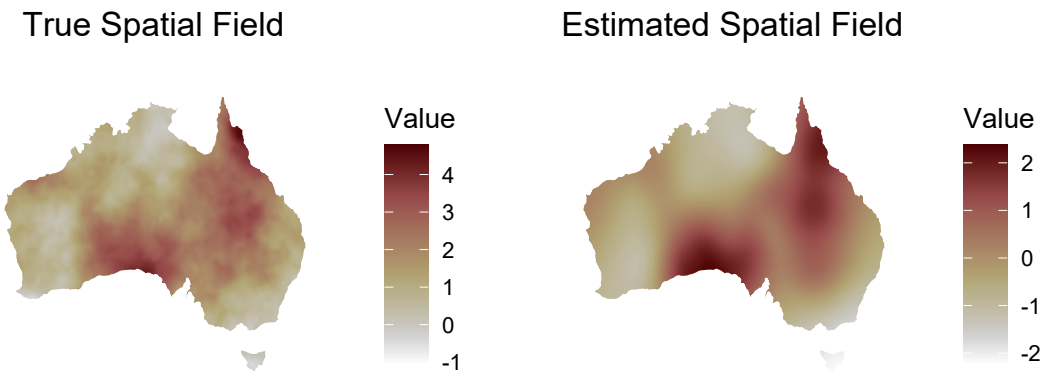
Okay, now we can try fitting the model:

```
spat_mod <- inla(resp ~ 0 + intercept +
                   f(spat_id, model = spde),
                 data = inla.stack.data(big_stack, spde = spde_fit),
                 control.predictor = list(A = inla.stack.A(big_stack), compute = TRUE,
                                          link = 1),
                 control.compute = list(config = TRUE),
                 num.threads = 10
                 )

summary(spat_mod)
```

```
##
## Call:
##    c("inla(formula = resp ~ 0 + intercept + f(spat_id, model = spde), ", "
##    data = inla.stack.data(big_stack, spde = spde_fit), control.compute =
##    list(config = TRUE), ", " control.predictor = list(A =
##    inla.stack.A(big_stack), compute = TRUE, ", " link = 1), num.threads =
##    10)")
## Time used:
##     Pre = 0.522, Running = 22.4, Post = 0.554, Total = 23.5
## Fixed effects:
##            mean   sd 0.025quant 0.5quant 0.975quant  mode kld
## intercept 1.518 0.03      1.458    1.518      1.578 1.517   0
##
## Random effects:
##   Name     Model
##     spat_id SPDE2 model
##
## Model hyperparameters:
##                                           mean       sd 0.025quant 0.5quant
## Precision for the Gaussian observations 18742.38 1.84e+04   1246.231 1.33e+04
## Range for spat_id                          17.61 4.92e+00     10.729 1.67e+01
## Stdev for spat_id                           1.01 1.91e-01      0.717 9.78e-01
##                                         0.975quant    mode
## Precision for the Gaussian observations   67504.82 3401.27
## Range for spat_id                            29.71   14.81
## Stdev for spat_id                             1.46    0.91
##
## Expected number of effective parameters(stdev): 39.92(0.107)
## Number of equivalent replicates : 1.00
##
## Marginal log-Likelihood:  -30.97
## Posterior marginals for the linear predictor and
##  the fitted values are computed
```

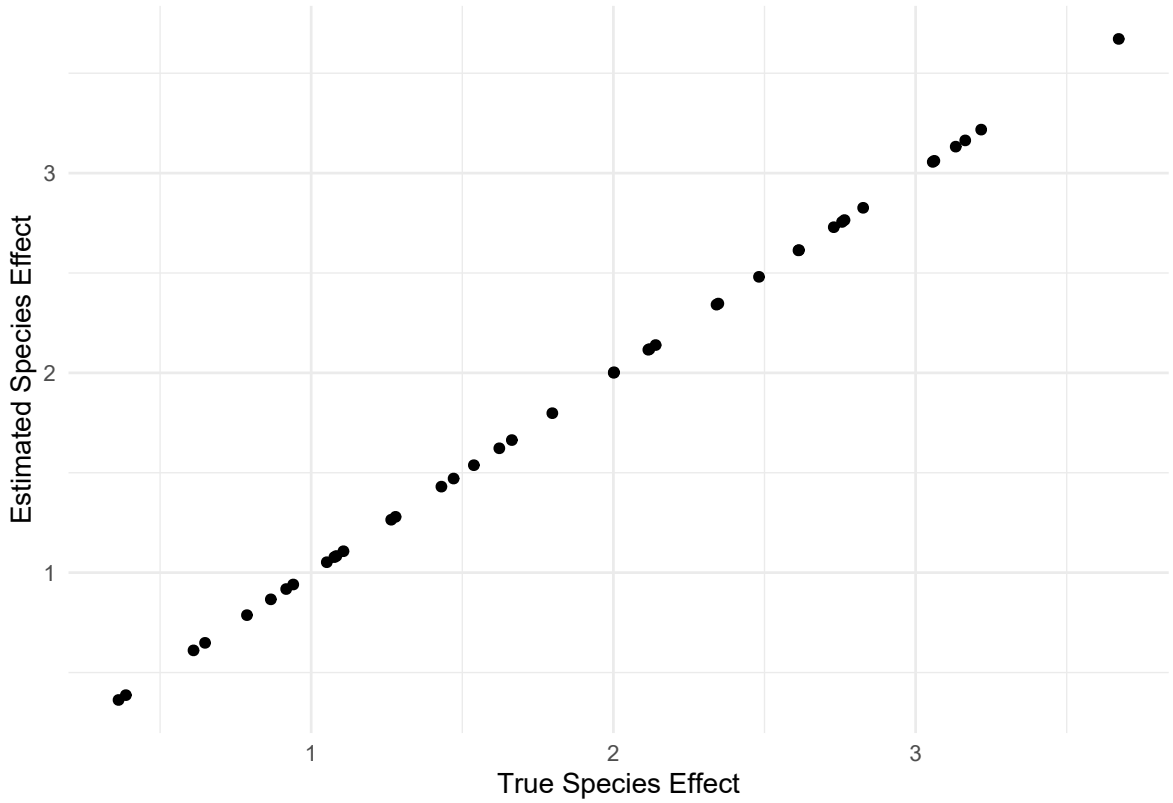
The first thing to note is that, incredibly, `INLA` has estimated spatial parameters very close to the simulated true values (though with a slight overestimate for the range – 18 instead of 15). This means that the 'discretization' induced by aggregating to the species level did not have a large effect on accuracy. In any case, we are more interested in whether the model has 'captured' the real spatial effect reasonably, which we will test qualititatively by plotting the true and the estimated spatial effect side by side.

```
{plot_spatial_field(u, spatial_mesh, aus_coast) + ggtitle("True Spatial Field") +
    theme(plot.title = element_text(hjust = 0.5))} +
  {plot_spatial_field(spat_mod$summary.ran$spat_id$mean, spatial_mesh, aus_coast) +
      ggtitle("Estimated Spatial Field") +
    theme(plot.title = element_text(hjust = 0.5))}
```



So that looks pretty good to me. The spatial effect has obviously been "smoothed" out somewhat. But that is expected, even without the species-level averaging, since the simulated data has noise added to it already in the `inla.qsample` function. Let's see how the model has done at estimating the species-level values.

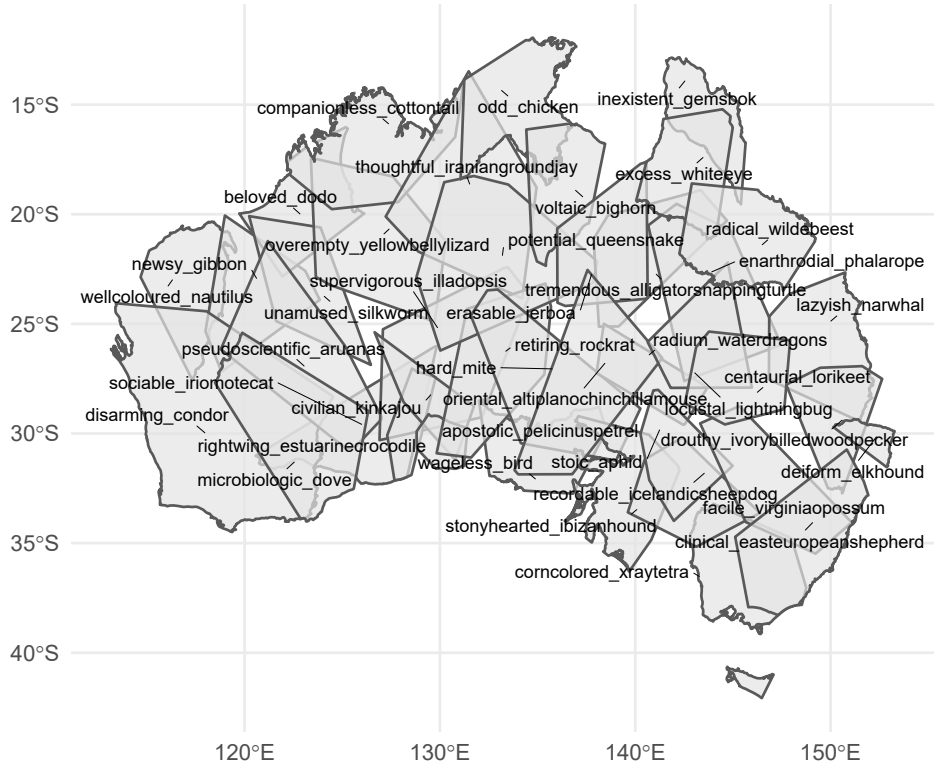
```
spat_index <- inla.stack.index(big_stack, "loc")
spec_spat_effect <- spat_mod$summary.fitted.values[spat_index$data, ]

ggplot(spec_df %>% dplyr::mutate(preds = spec_spat_effect$mean +
                                   spat_mod$summary.fixed$mean), aes(u, preds)) +
  geom_point() +
  ylab("Estimated Species Effect") +
  xlab("True Species Effect") +
  theme_minimal()
```



So that has done a nearly perfect job of estimating the species-level effect. This is perhaps not surprising given we have given each species a pretty good sample size. Now let's try expanding our species' ranges and/or scaling and/or rotating them so that their borders overlap. We'll write a function to do this easily.

```
scale_vec <- runif(n_spec, 1, 1.5)
translate_list <- replicate(n_spec, runif(2, -1, 1), simplify = FALSE)
rotate_vec <- runif(n_spec, 0, 360)

affine_transform_ranges <- function(ranges, scale_vec, translate_list, rotate_vec, aus_coast) {
  rotate <- rotate_vec * pi / 180
  rot <- function(a) matrix(c(cos(a), sin(a), -sin(a), cos(a)), 2, 2)

  rot_list <- lapply(rotate, rot)

  suppressWarnings({cnts <- sf::st_centroid(ranges) %>% sf::st_geometry();
  ranges2 <- sf::st_geometry(ranges)})

  ranges2 <- (ranges2 - cnts) * rot_list * scale_vec + cnts + translate_list

  ranges2 <- ranges2 %>%
    sf::st_sf() %>%
    sf::st_set_crs(4326) %>%
    sf::st_intersection(., aus_coast %>% dplyr::select(geometry)) %>%
    sf::st_cast() %>%
    dplyr::mutate(species = ranges$species[1:n()])

  ranges2
}
```

```
new_ranges <- affine_transform_ranges(range_sample, scale_vec, translate_list, rotate_vec, aus_coast)

ggplot(new_ranges) +
  geom_sf(alpha = 0.7) +
  ggrepel::geom_text_repel(
      aes(label = species, geometry = geometry),
      data = range_sample,
      stat = "sf_coordinates",
      min.segment.length = 0,
      segment.size = 0.1,
      size = 2.5,
      box.padding = 0.25,
      force = 2,
      max.iter = 5000
    ) +
  xlab("") +
  ylab("") +
  theme_minimal()
```

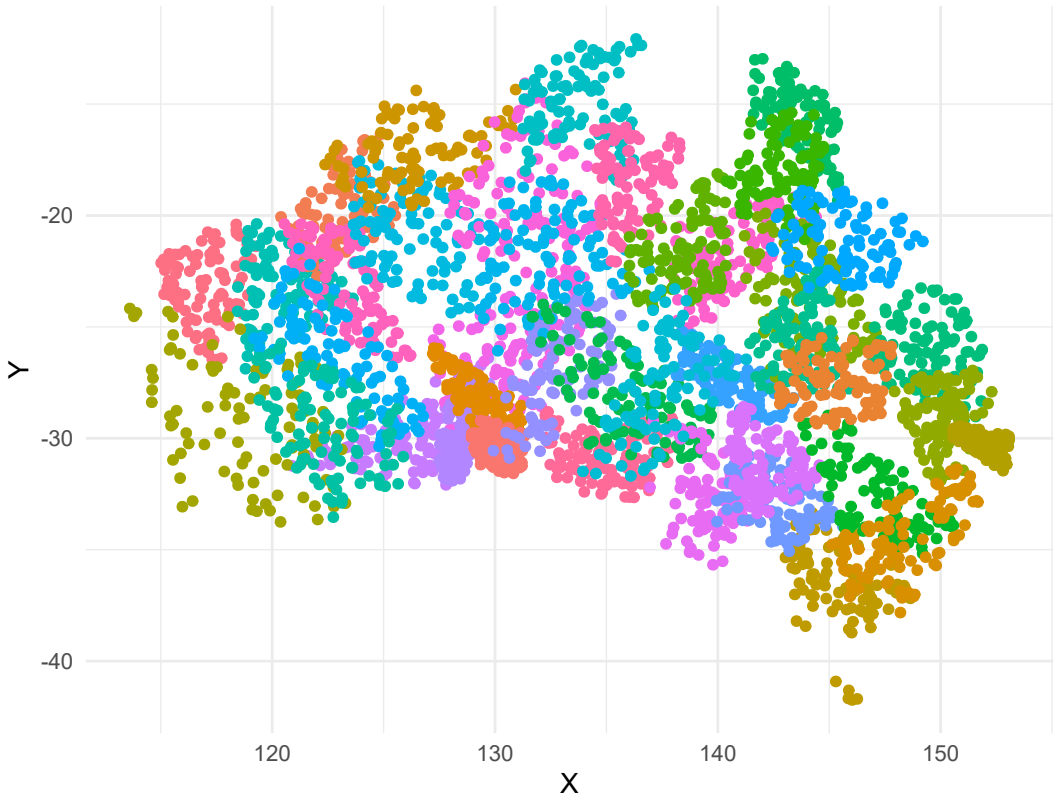


Okay cool, now lot's of overlap. Let's wrap all the code for sampling from the ranges and then fitting the model into a function, and then run it on our new overlapping ranges.

```
sample_and_fit <- function(n_pnt_per_spec, noise_sd, ranges, spatial_mesh, u) {
  n_spec <- length(ranges$species)
  size <- rep(n_pnt_per_spec, n_spec)
  suppressMessages(point_sample <- ranges %>%
```

```
  sf::st_sample(size = size) %>%
  sf::st_sf(ID = rep(seq_along(size), size),
            geometry = .) %>%
  dplyr::mutate(species = ranges$species[ID]))

pnt_df <- point_sample %>%
  sf::st_coordinates() %>%
  dplyr::as_tibble() %>%
  dplyr::mutate(species = point_sample$species)

## project to mesh
proj <- inla.mesh.projector(spatial_mesh, loc = as.matrix(pnt_df %>% dplyr::select(X, Y)))
u_proj <- inla.mesh.project(proj, u)

pnt_df <- pnt_df %>%
  dplyr::mutate(u = u_proj)

spec_df <- pnt_df %>%
  dplyr::group_by(species) %>%
  dplyr::summarise(u = mean(u)) %>%
  dplyr::mutate(u = u + rnorm(n(), sd = noise_sd)) ## add species-level noise


## use the same priors we used in the manuscript
spde_fit <- inla.spde2.pcmatern(
      mesh = spatial_mesh, alpha = 2, ### mesh and smoothness parameter
      prior.range = c(2, 0.1), ### P(practic.range<2)=0.1
      prior.sigma = c(1, 0.1),
      constr = TRUE)

coords <- as.matrix(pnt_df[ , c("X", "Y")])
A <- inla.spde.make.A(spatial_mesh, loc = coords) ## this creates a sparse matrix
dim(A)

specs <- spec_df$species

spec_A_rows <- lapply(specs, function(x) { ## apply to each species one at a time

  A[which(pnt_df$species == x), ] %>% ## get rows associated with the species
    apply(2, mean) ## average each column to return a single row with the same number of columns

}) %>%
  do.call(rbind, .) ## rbind the species back together into one matrix

## convert back to sparse matrix
spec_A_rows <- Matrix(spec_A_rows, sparse = TRUE)

## check some stuff
dim(spec_A_rows)
apply(spec_A_rows, 1, sum) %>% head

stack <- inla.stack(data = list(resp = spec_df$u),
                    A = list(spec_A_rows, 1), ## one A for each effect, 1 is shorthand, see below
```

```
                          effects=list(spat_id = 1:spde_fit$n.spde,
                                       intercept = rep(1, nrow(spec_df))),
                          tag='est')

  stck.loc <- inla.stack(data = list(resp = NA),
                         A = list(spec_A_rows),
                         effects = list(spat_id = 1:spde_fit$n.spde), tag='loc')

  big_stack <- inla.stack(stack, stck.loc)

  spat_mod <- inla(resp ~ 0 + intercept +
                   f(spat_id, model = spde),
                 data = inla.stack.data(big_stack, spde = spde_fit),
                 control.predictor = list(A = inla.stack.A(big_stack), compute = TRUE,
                                          link = 1),
                 control.compute = list(config = TRUE),
                 num.threads = 10
                 )

  list(spec_df = spec_df, pnt_df = pnt_df, mod = spat_mod, stack = big_stack)

}

new_fit <- sample_and_fit(n_pnt_per_spec, noise_sd, new_ranges, spatial_mesh, u)

ggplot(new_fit$pnt_df, aes(X, Y)) +
  geom_point(aes(colour = species)) +
  coord_equal() +
  theme_minimal() +
  theme(legend.position = "none")
```

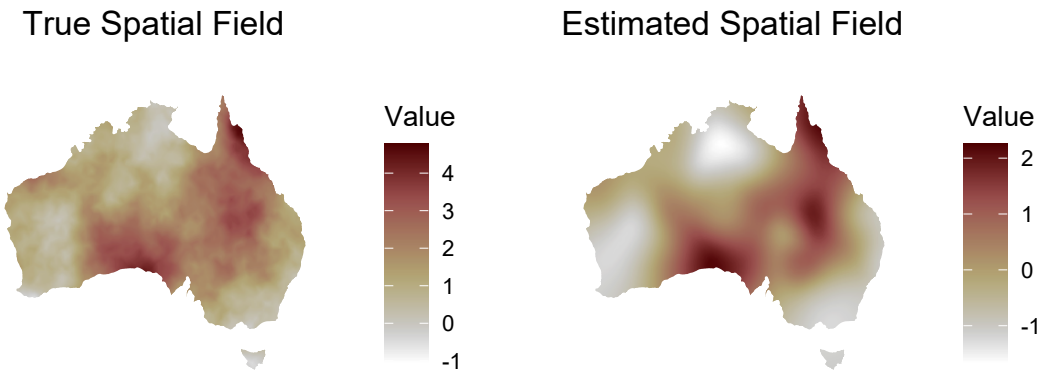



```
summary(new_fit$mod)
```

```
##
## Call:
##    c("inla(formula = resp ~ 0 + intercept + f(spat_id, model = spde), ", "
##    data = inla.stack.data(big_stack, spde = spde_fit), control.compute =
##    list(config = TRUE), ", " control.predictor = list(A =
##    inla.stack.A(big_stack), compute = TRUE, ", " link = 1), num.threads =
##    10)")
## Time used:
##     Pre = 0.402, Running = 32.4, Post = 0.413, Total = 33.2
## Fixed effects:
##             mean    sd 0.025quant 0.5quant 0.975quant  mode kld
## intercept 1.614 0.064      1.488    1.614      1.741 1.613   0
##
## Random effects:
##   Name     Model
##     spat_id SPDE2 model
##
## Model hyperparameters:
##                                           mean       sd 0.025quant 0.5quant
## Precision for the Gaussian observations 1.86e+04 1.83e+04   1199.424 1.31e+04
## Range for spat_id                       1.23e+01 2.99e+00      7.826 1.18e+01
## Stdev for spat_id                       9.73e-01 1.46e-01      0.736 9.56e-01
##                                         0.975quant     mode
## Precision for the Gaussian observations   67221.57 3239.905
## Range for spat_id                            19.47   10.778
```

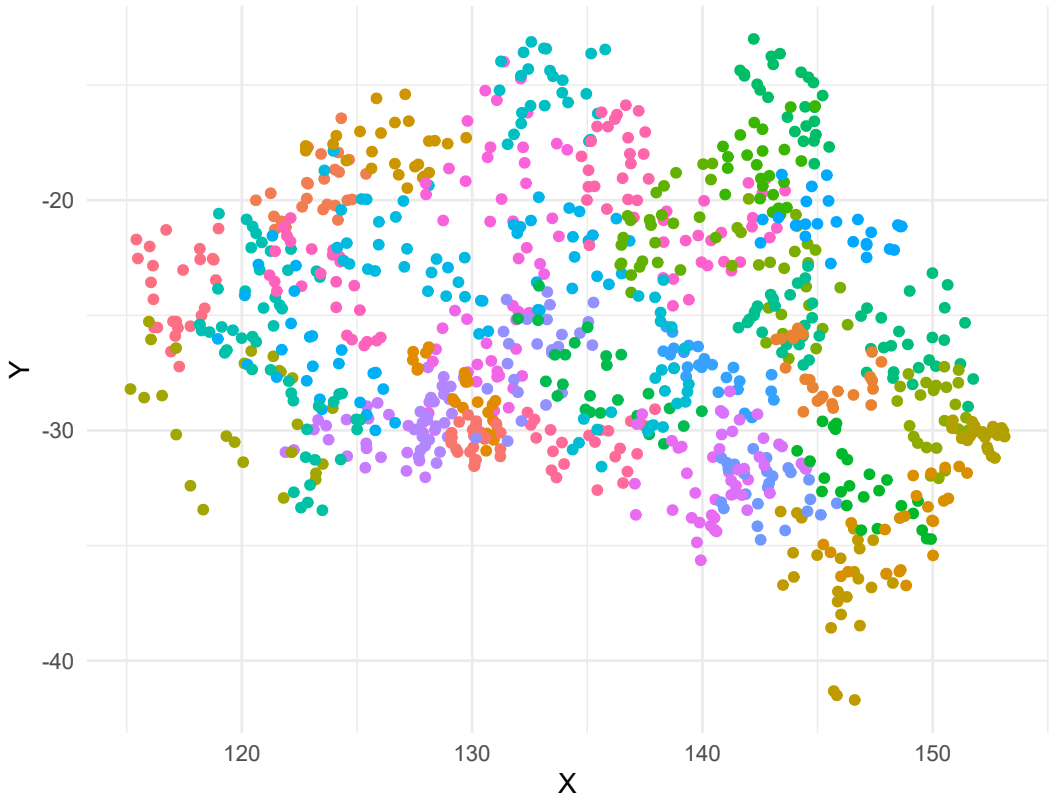
```
## Stdev for spat_id                                1.31    0.915
##
## Expected number of effective parameters(stdev): 39.86(0.195)
## Number of equivalent replicates : 1.00
##
## Marginal log-Likelihood:  -26.36
## Posterior marginals for the linear predictor and
##  the fitted values are computed
```

```
{plot_spatial_field(u, spatial_mesh, aus_coast) + ggtitle("True Spatial Field") +
    theme(plot.title = element_text(hjust = 0.5))} +
  {plot_spatial_field(new_fit$mod$summary.ran$spat_id$mean, spatial_mesh, aus_coast) +
      ggtitle("Estimated Spatial Field") +
    theme(plot.title = element_text(hjust = 0.5))}
```



What happens if we use few points per species?

```
new_fit2 <- sample_and_fit(25, noise_sd, new_ranges, spatial_mesh, u)

ggplot(new_fit2$pnt_df, aes(X, Y)) +
  geom_point(aes(colour = species)) +
  coord_equal() +
  theme_minimal() +
  theme(legend.position = "none")
```

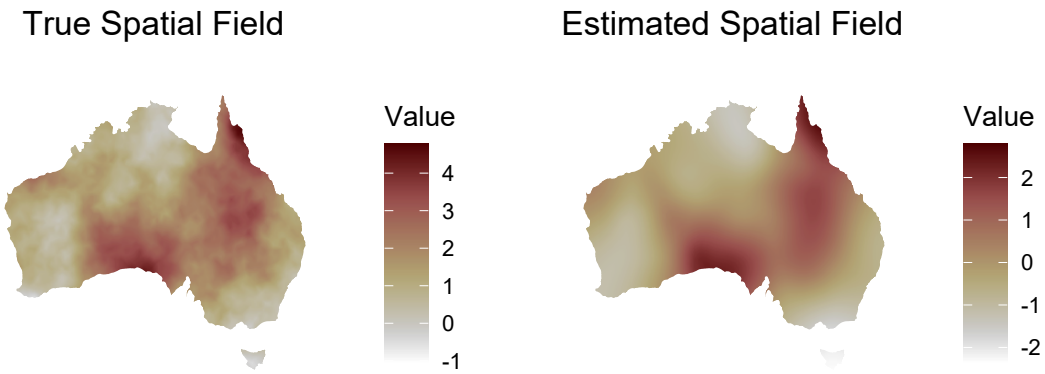



```
summary(new_fit2$mod)
```

```
##
## Call:
##    c("inla(formula = resp ~ 0 + intercept + f(spat_id, model = spde), ", "
##    data = inla.stack.data(big_stack, spde = spde_fit), control.compute =
##    list(config = TRUE), ", " control.predictor = list(A =
##    inla.stack.A(big_stack), compute = TRUE, ", " link = 1), num.threads =
##    10)")
## Time used:
##     Pre = 0.37, Running = 11.4, Post = 0.314, Total = 12.1
## Fixed effects:
##             mean    sd 0.025quant 0.5quant 0.975quant  mode kld
## intercept 1.537 0.056      1.428    1.537      1.649 1.536   0
##
## Random effects:
##   Name     Model
##     spat_id SPDE2 model
##
## Model hyperparameters:
##                                           mean       sd 0.025quant 0.5quant
## Precision for the Gaussian observations 18724.56 1.84e+04   1304.709 13310.46
## Range for spat_id                          18.79 5.42e+00     11.243    17.73
## Stdev for spat_id                           1.09 2.21e-01      0.759     1.06
##                                         0.975quant     mode
## Precision for the Gaussian observations   67325.34 3579.391
## Range for spat_id                            32.17   15.732
```

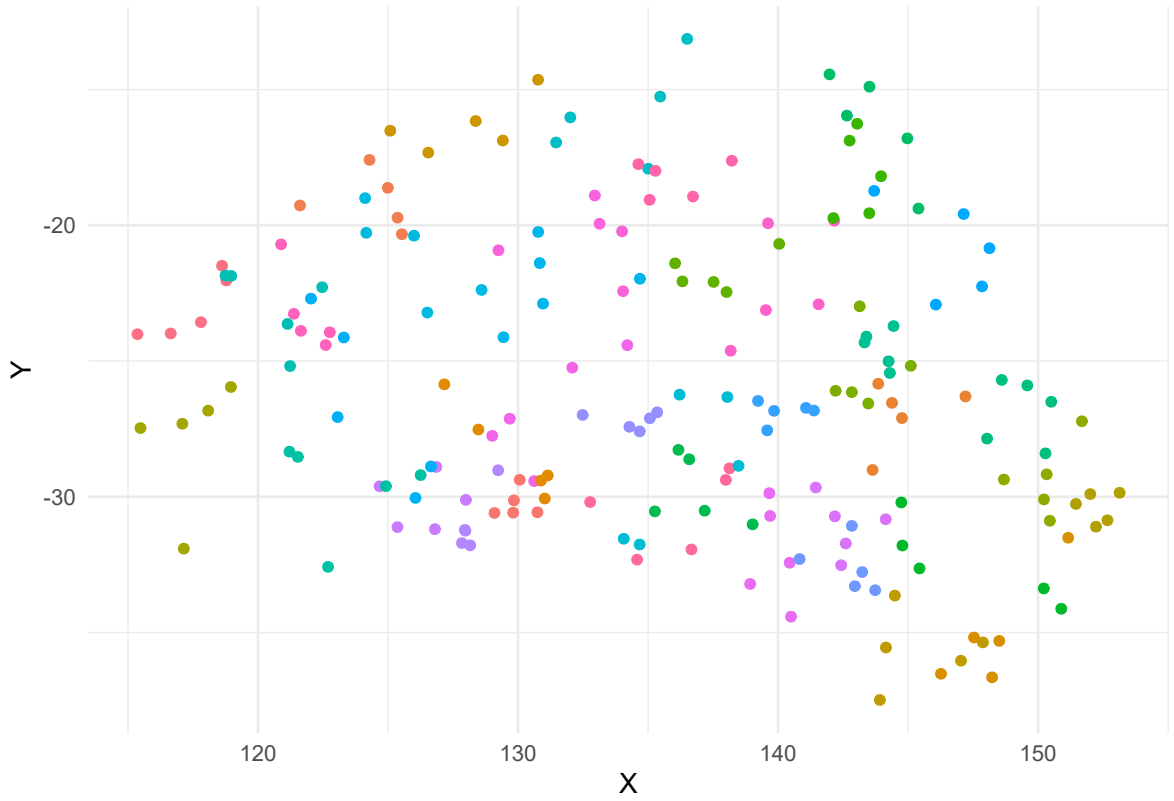
```
## Stdev for spat_id                                1.62    0.979
##
## Expected number of effective parameters(stdev): 39.81(0.251)
## Number of equivalent replicates : 1.00
##
## Marginal log-Likelihood:  -21.86
## Posterior marginals for the linear predictor and
##  the fitted values are computed
```

```
{plot_spatial_field(u, spatial_mesh, aus_coast) + ggtitle("True Spatial Field") +
    theme(plot.title = element_text(hjust = 0.5))} +
  {plot_spatial_field(new_fit2$mod$summary.ran$spat_id$mean, spatial_mesh, aus_coast) +
      ggtitle("Estimated Spatial Field") +
    theme(plot.title = element_text(hjust = 0.5))}
```



The sample is considerably sparser but we are still getting a pretty good estimate. How low can we go?

```
new_fit3 <- sample_and_fit(5, noise_sd, new_ranges, spatial_mesh, u)

ggplot(new_fit3$pnt_df, aes(X, Y)) +
  geom_point(aes(colour = species)) +
  theme_minimal() +
  theme(legend.position = "none")
```

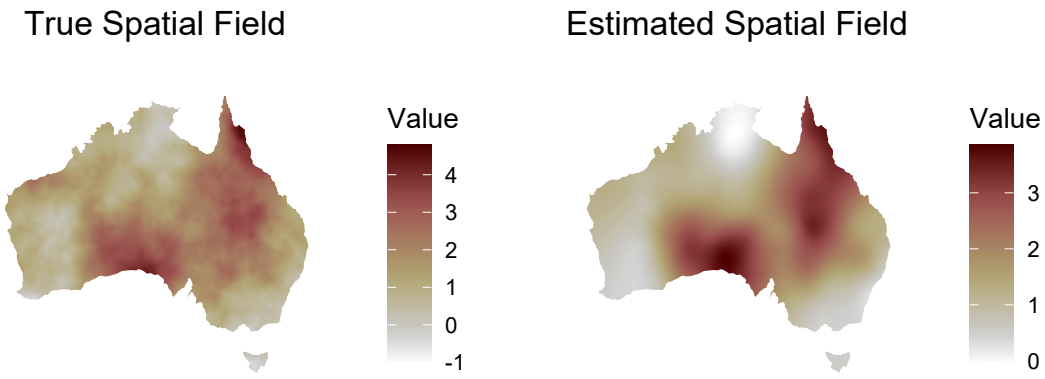



```
summary(new_fit3$mod)
```

```
##
## Call:
##    c("inla(formula = resp ~ 0 + intercept + f(spat_id, model = spde), ", "
##    data = inla.stack.data(big_stack, spde = spde_fit), control.compute =
##    list(config = TRUE), ", " control.predictor = list(A =
##    inla.stack.A(big_stack), compute = TRUE, ", " link = 1), num.threads =
##    10)")
## Time used:
##     Pre = 0.376, Running = 7.61, Post = 0.247, Total = 8.23
## Fixed effects:
##             mean    sd 0.025quant 0.5quant 0.975quant  mode kld
## intercept 1.554 0.085      1.386    1.554      1.721 1.554   0
##
## Random effects:
##   Name     Model
##     spat_id SPDE2 model
##
## Model hyperparameters:
##                                           mean       sd 0.025quant 0.5quant
## Precision for the Gaussian observations 18449.96 1.83e+04   1242.820 1.30e+04
## Range for spat_id                          14.14 3.66e+00      8.789 1.35e+01
## Stdev for spat_id                           1.02 1.76e-01      0.743 9.98e-01
##                                         0.975quant     mode
## Precision for the Gaussian observations   66858.07 3385.274
## Range for spat_id                            23.02   12.233
```

```
## Stdev for spat_id                                1.43    0.943
##
## Expected number of effective parameters(stdev): 39.91(0.119)
## Number of equivalent replicates : 1.00
##
## Marginal log-Likelihood:  -30.60
## Posterior marginals for the linear predictor and
##  the fitted values are computed
```

```
{plot_spatial_field(u, spatial_mesh, aus_coast) + ggtitle("True Spatial Field") +
    theme(plot.title = element_text(hjust = 0.5))} +
  {plot_spatial_field(new_fit3$mod$summary.ran$spat_id$mean + new_fit3$mod$summary.fixed$mean,
                      spatial_mesh, aus_coast) +
      ggtitle("Estimated Spatial Field") +
    theme(plot.title = element_text(hjust = 0.5))}
```

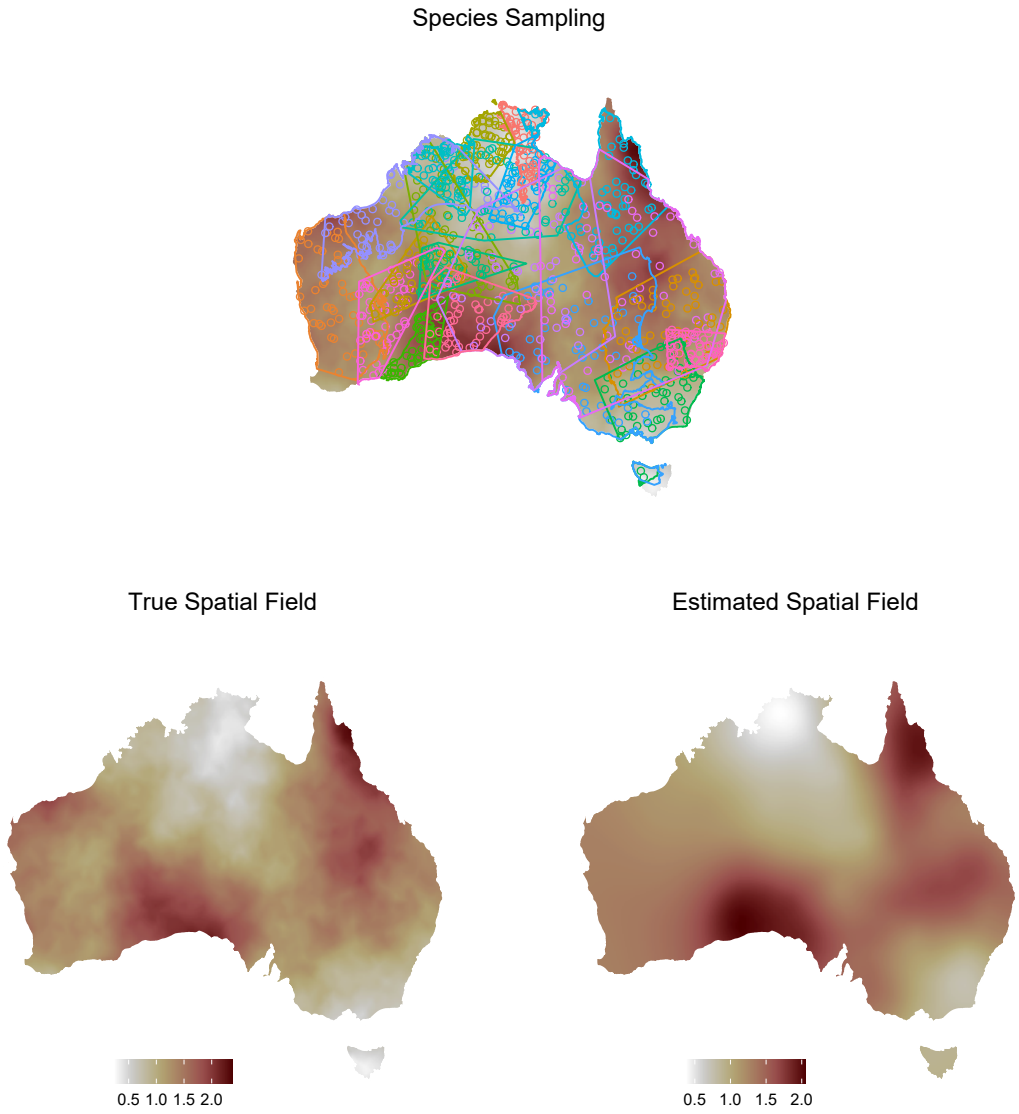


Even with only 5 points per species we are still doing pretty well. Perhaps it is really the number of species that matters more? Let's write a function to run the full simulation, fit the model, then return summaries including plots.

```
sim_and_fit_spatial <- function(n_spec = 20, range = 24, sigma.u = 0.5, n_pnt_per_spec = 50L,
                                noise_sd = 0.1, scale_min = 1, scale_max = 2,
                                translate_min = -1, translate_max = 1, rotate_min = 0,
                                rotate_max = 360, spatial_mesh, aus_coast) {

  ## This function requires specifying a prior, although they are not used for this application
  spde = inla.spde2.pcmatern(spatial_mesh, prior.range = c(.5, .5), prior.sigma = c(.5, .5))
```

```r
Qu = inla.spde.precision(spde, theta=c(log(range), log(sigma.u)))
u = inla.qsample(n = 1, Q = Qu, seed = inla.seed)

if(nrow(u) > 1) {
  u <- u[ , 1]
}

range_sample <- aus_coast %>%
  sf::st_sample(n_spec) %>%
  sf::st_combine() %>%
  sf::st_voronoi(envelope = aus_coast$geometry[[1]]) %>%
  sf::st_cast() %>%
  data.frame(geometry = .) %>%
  sf::st_sf(.) %>%
  sf::st_transform(4326) %>%
  sf::st_intersection(., aus_coast %>% dplyr::select(geometry)) %>%
  sf::st_cast() %>%
  dplyr::mutate(species = ids::adjective_animal(n = n()))

scale_vec <- runif(n_spec, scale_min, scale_max)
translate_list <- replicate(n_spec, runif(2, translate_min, translate_max), simplify = FALSE)
rotate_vec <- runif(n_spec, rotate_min, rotate_max)

new_ranges <- affine_transform_ranges(range_sample, scale_vec, translate_list, rotate_vec, aus_coast)

fit <- sample_and_fit(n_pnt_per_spec, noise_sd, new_ranges, spatial_mesh, u)

sample_plot <- {plot_spatial_field(u, spatial_mesh, aus_coast) +
    geom_point(aes(X, Y, colour = species), shape = 21, data = fit$pnt_df) +
    geom_sf(aes(geometry = geometry, colour = species), fill = NA, data = new_ranges, inherit.aes = F
    scale_colour_discrete(guide = "none") +
    ggtitle("Species Sampling") +
    theme(plot.title = element_text(hjust = 0.5),
          legend.position = "none")}

compare_plot <- {plot_spatial_field(u, spatial_mesh, aus_coast) +
    ggtitle("True Spatial Field") +
    theme(plot.title = element_text(hjust = 0.5),
        legend.direction = "horizontal",
        legend.position = c(0.4, 0.1),
        legend.key.size = unit(0.02, "npc"),
        legend.title = element_blank(),
        plot.margin = unit(c(0, 0, 0, 0), "npc"))} +
{plot_spatial_field(fit$mod$summary.ran$spat_id$mean + fit$mod$summary.fixed$mean,
                    spatial_mesh, aus_coast) +
    ggtitle("Estimated Spatial Field") +
  theme(plot.title = element_text(hjust = 0.5),
        legend.direction = "horizontal",
        legend.position = c(0.4, 0.1),
        legend.key.size = unit(0.02, "npc"),
        legend.title = element_blank(),
        plot.margin = unit(c(0, 0, 0, 0), "npc"))}
```

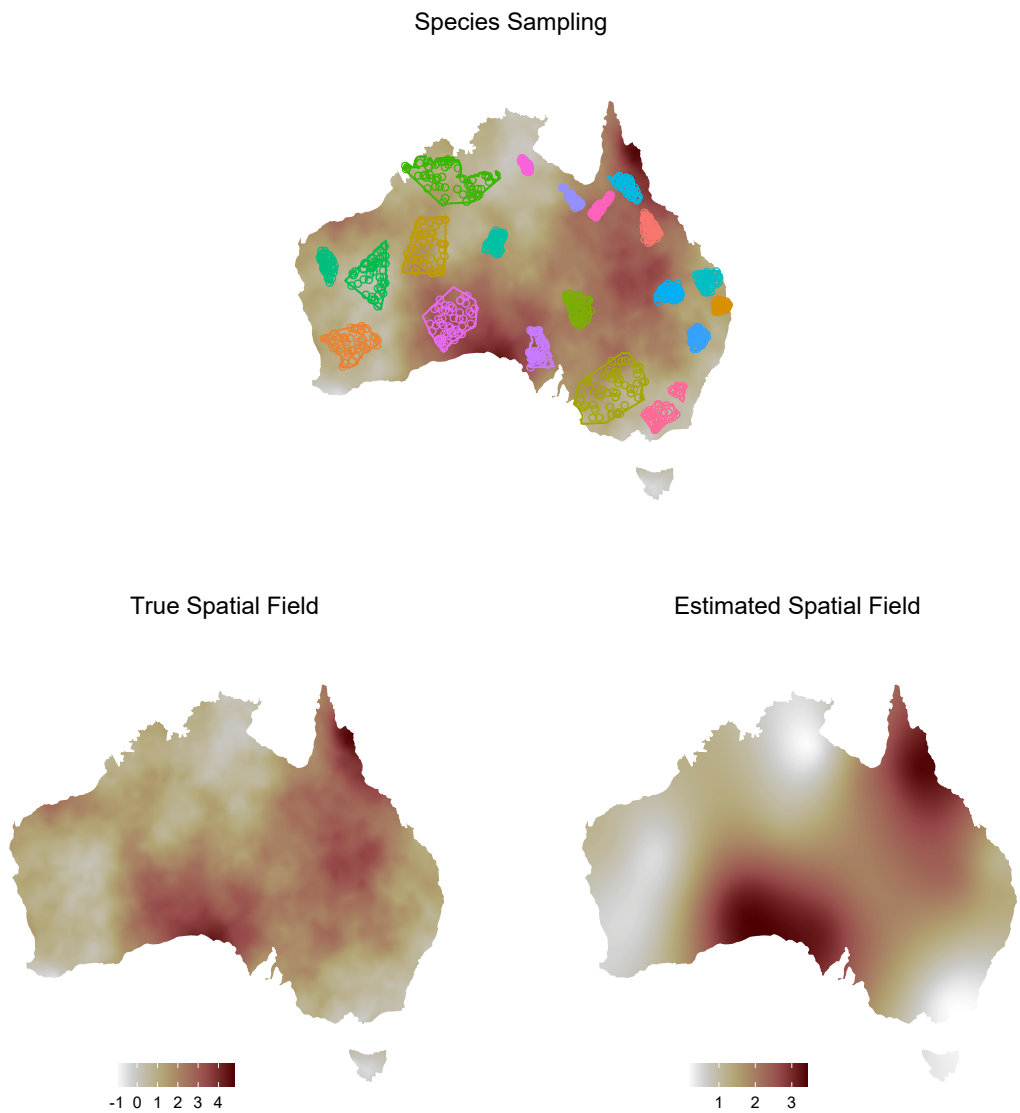
```
  combined_plot <- sample_plot + compare_plot + plot_layout(ncol = 1, heights = c(1, 1))

  list(fit = fit, u = u, compare_plot = compare_plot, sample_plot = sample_plot, combined_plot = combin
}

test_sim <- sim_and_fit_spatial(spatial_mesh = spatial_mesh, aus_coast = aus_coast)

test_sim$combined_plot
```



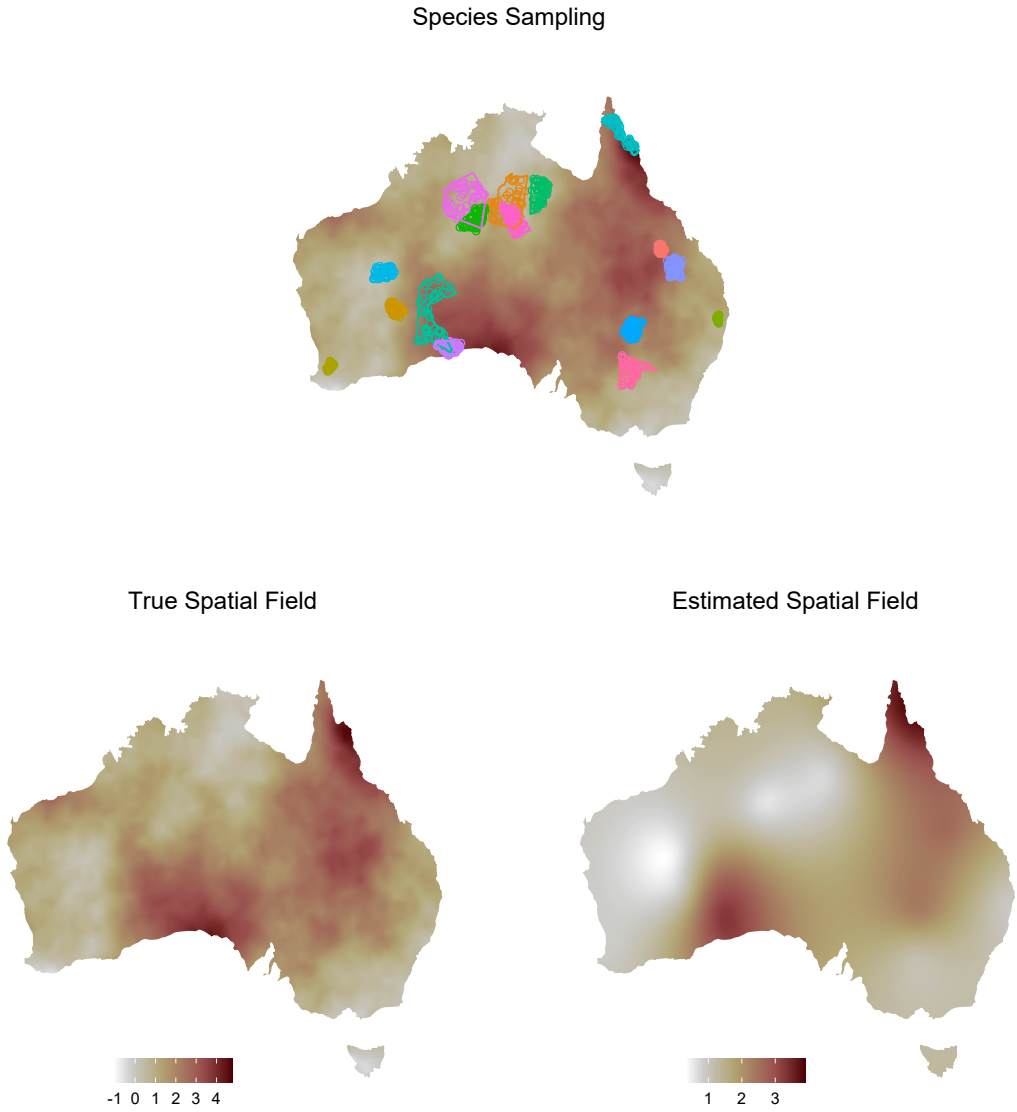
The emerging picture here is that the model seems to do a pretty good job almost regardless of the number of samples or species, at least with spatial effects on the sort of scale we are testing here. There is two situations where we might expect the model to do worse. The first is when the species don't cover the area of the spatial effect very well. The second is when the spatial effect are very fine such that the range of the effect is less than the average size of a species's range. Then the discretization created by aggregating at the species-level will make detecting such fine scale nearly impossible. In this situation we might expect species overlap to help, because the model will have more fine scale information since each 'overlap' zone the model

will be able to estimate a different effect. Let's try some of these scenarios.

```
sim_low_cover <- sim_and_fit_spatial(range = 15, sigma.u = 1, scale_min = 0.1, scale_max = 0.7,
                                     spatial_mesh = spatial_mesh, aus_coast = aus_coast)

sim_low_cover$combined_plot
```

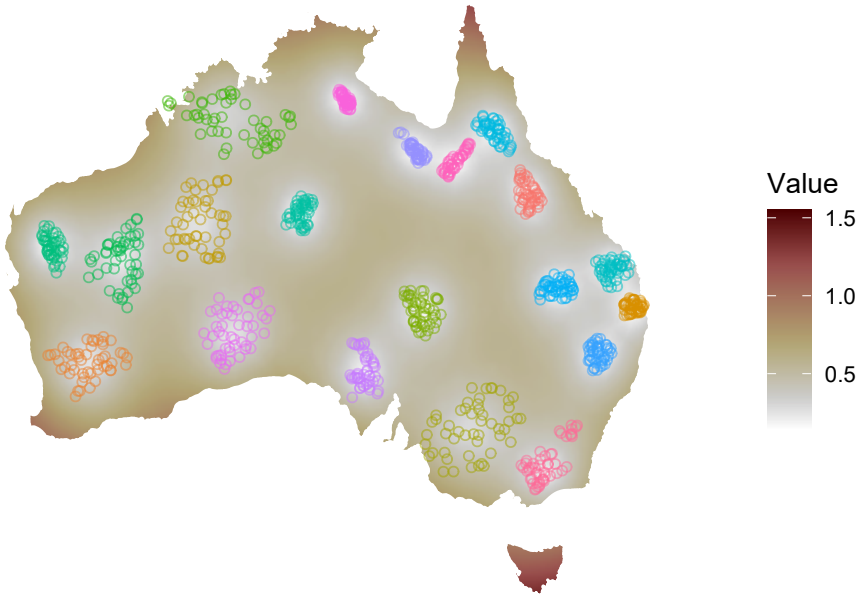



```
sim_low_cover2 <- sim_and_fit_spatial(range = 15, sigma.u = 1, scale_min = 0.1, scale_max = 0.7,
                                      translate_min = -20, translate_max = 20, n_spec = 30,
                                     spatial_mesh = spatial_mesh, aus_coast = aus_coast)

sim_low_cover2$combined_plot
```

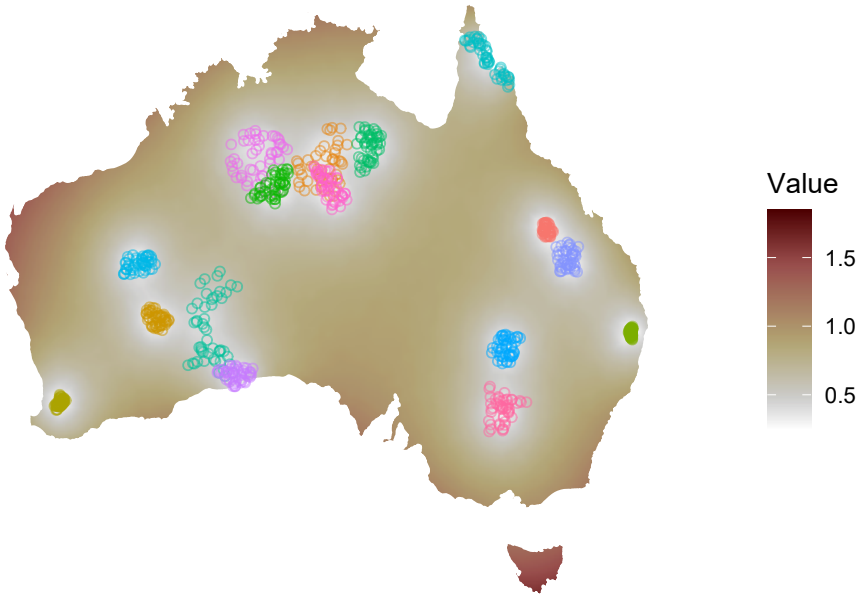


We are still pretty accurate if the species ranges are relatively uniformly distributed in the space, but if they are more clustered we start to see less accuracy. We can also look at the uncertainty in our spatial estimates:

```
plot_spatial_field(sim_low_cover$fit$mod$summary.random$spat_id$sd, spatial_mesh, aus_coast) +
  geom_point(aes(X, Y, colour = species), shape = 21, data = sim_low_cover$fit$pnt_df, alpha = 0.5) +
  scale_colour_discrete(guide = "none")
```




```
plot_spatial_field(sim_low_cover2$fit$mod$summary.random$spat_id$sd, spatial_mesh, aus_coast) +
  geom_point(aes(X, Y, colour = species), shape = 21, data = sim_low_cover2$fit$pnt_df, alpha = 0.5) +
  scale_colour_discrete(guide = "none")
```



We have high uncertainty (as measured by the standard deviation of the posterior distribution) where there are few points, as we would expect.

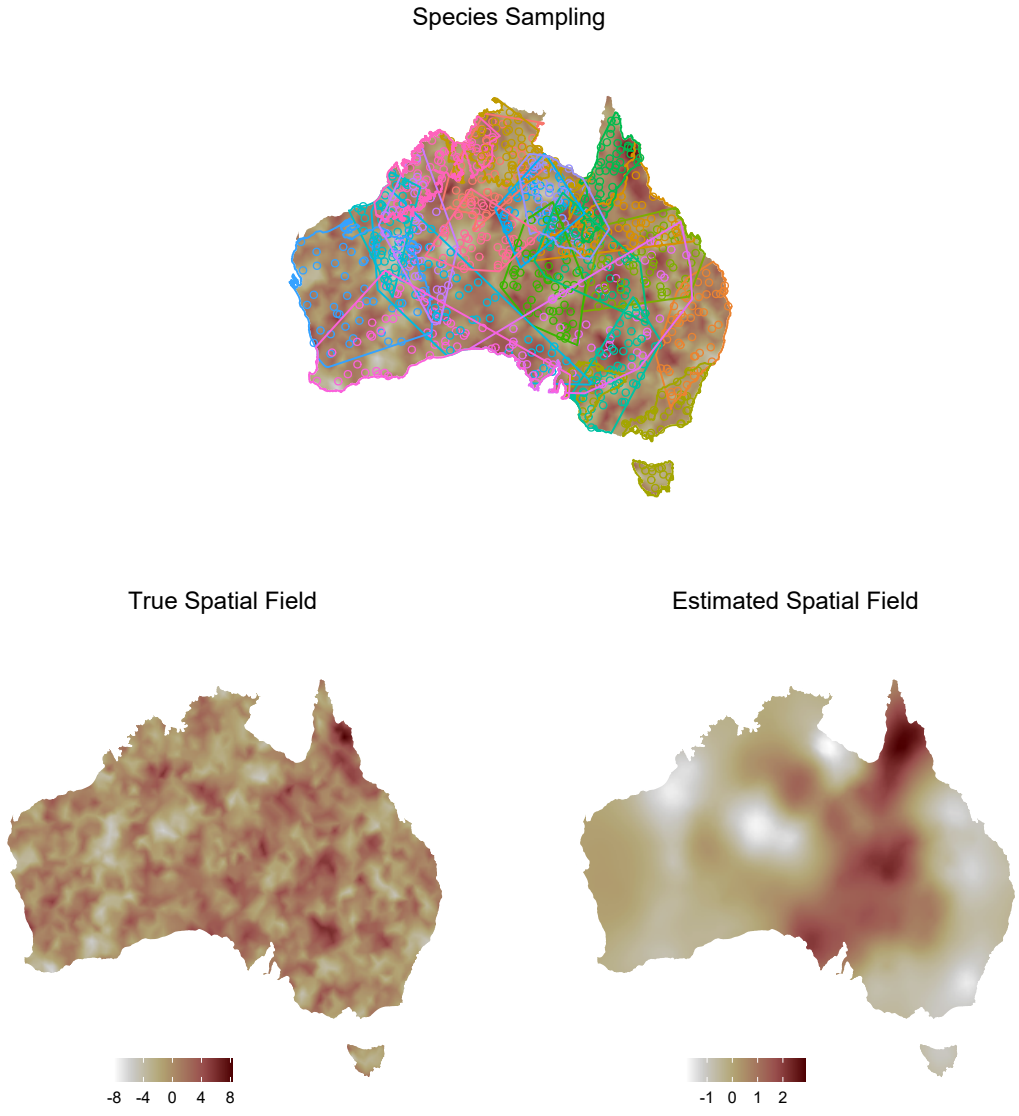
Now let's try some fine-scale spatial patterns.

```
sim_fine <- sim_and_fit_spatial(range = 2, sigma.u = 2, spatial_mesh = spatial_mesh, aus_coast = aus_co

summary(sim_fine$fit$mod)
```

```
##
## Call:
##    c("inla(formula = resp ~ 0 + intercept + f(spat_id, model = spde), ", "
##    data = inla.stack.data(big_stack, spde = spde_fit), control.compute =
##    list(config = TRUE), ", " control.predictor = list(A =
##    inla.stack.A(big_stack), compute = TRUE, ", " link = 1), num.threads =
##    10)")
## Time used:
##     Pre = 0.372, Running = 17.4, Post = 0.394, Total = 18.2
## Fixed effects:
##             mean    sd 0.025quant 0.5quant 0.975quant   mode kld
## intercept -0.006 0.138     -0.281   -0.006      0.266 -0.005   0
##
## Random effects:
##   Name     Model
##     spat_id SPDE2 model
##
## Model hyperparameters:
##                                          mean       sd 0.025quant 0.5quant
```

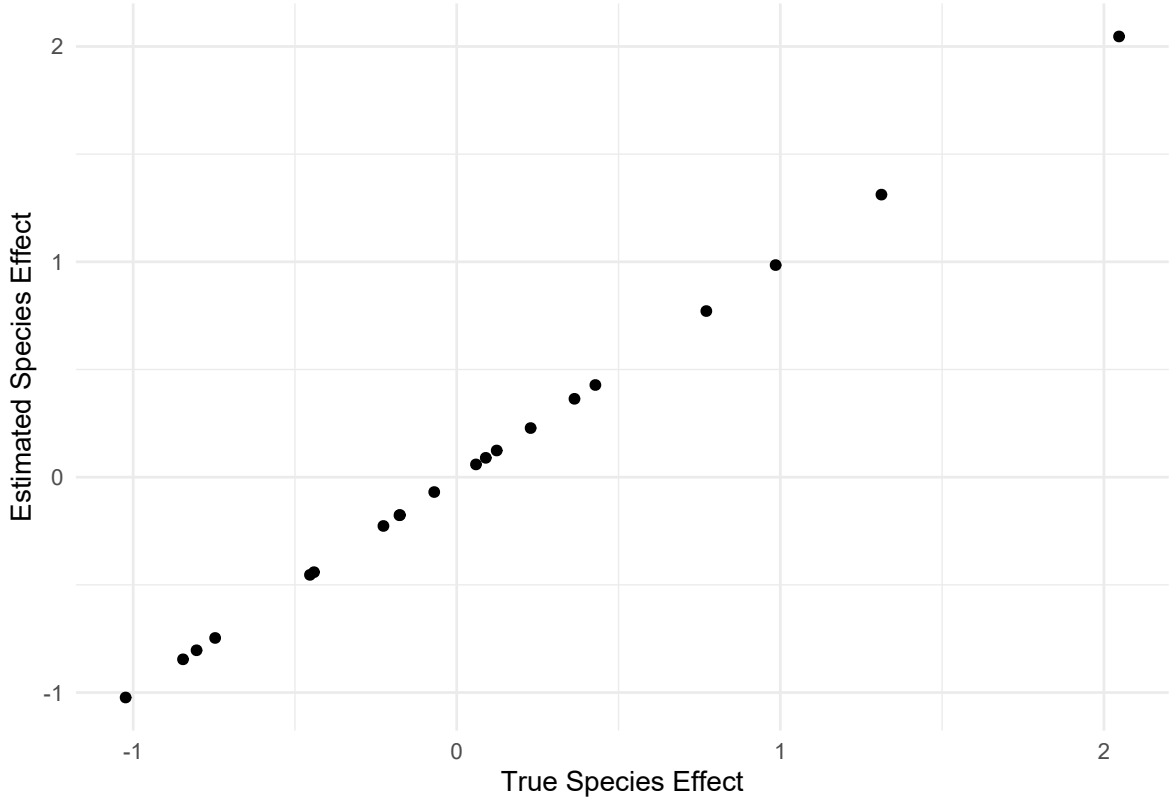
```
## Precision for the Gaussian observations 18586.11 1.83e+04    1249.86 13164.78
## Range for spat_id                           3.21 1.12e+00       1.46     3.08
## Stdev for spat_id                           1.96 3.56e-01       1.38     1.91
##                                          0.975quant    mode
## Precision for the Gaussian observations   67143.17 3405.77
## Range for spat_id                             5.81    2.80
## Stdev for spat_id                             2.77    1.81
##
## Expected number of effective parameters(stdev): 19.99(0.012)
## Number of equivalent replicates : 1.00
##
## Marginal log-Likelihood:  -30.75
## Posterior marginals for the linear predictor and
##  the fitted values are computed
```

```
sim_fine$combined_plot
```



Ah, here we see where the model starts to fall down. It has overestimated the range parameter and underestimated the standard deviation of the spatial effect. Plotting the spatial effect estimate shows that the model has picked up a larger-scale pattern that looks not incorrect, which is probably the result of some random fluctuations at a higher spatial scale than the scale on which the data has been generated. On the other hand the credible intervals of the spatial parameters do contain the true values. What about the species-level effect?

```
spat_index <- inla.stack.index(sim_fine$fit$stack, "loc")
spec_spat_effect <- sim_fine$fit$mod$summary.fitted.values[spat_index$data, ]

ggplot(sim_fine$fit$spec_df %>% dplyr::mutate(preds = spec_spat_effect$mean +
                                  sim_fine$fit$mod$summary.fixed$mean), aes(u, preds)) +
  geom_point() +
  ylab("Estimated Species Effect") +
  xlab("True Species Effect") +
  theme_minimal()
```



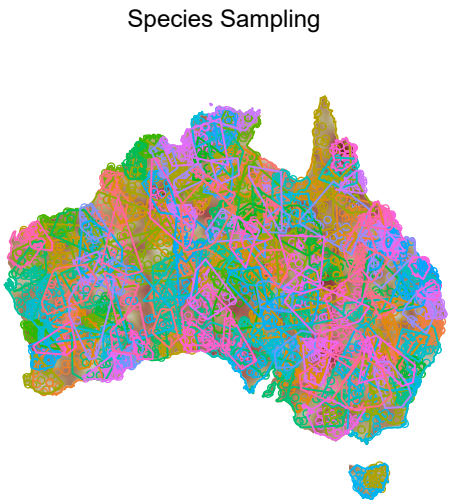
That still looks pretty decent, meaning that the model is still doing a good job aggregating across the whole ranges of the species to estimate the mean spatial effect. Can we pick up more of the fine scale spatial pattern if we have more species with smaller ranges?

```
sim_fine2 <- sim_and_fit_spatial(n_spec = 200, range = 2, sigma.u = 2,
                                 n_pnt_per_spec = 20, translate_min = -0.05, translate_max = 0.05,
                                 spatial_mesh = spatial_mesh, aus_coast = aus_coast)

summary(sim_fine2$fit$mod)
```

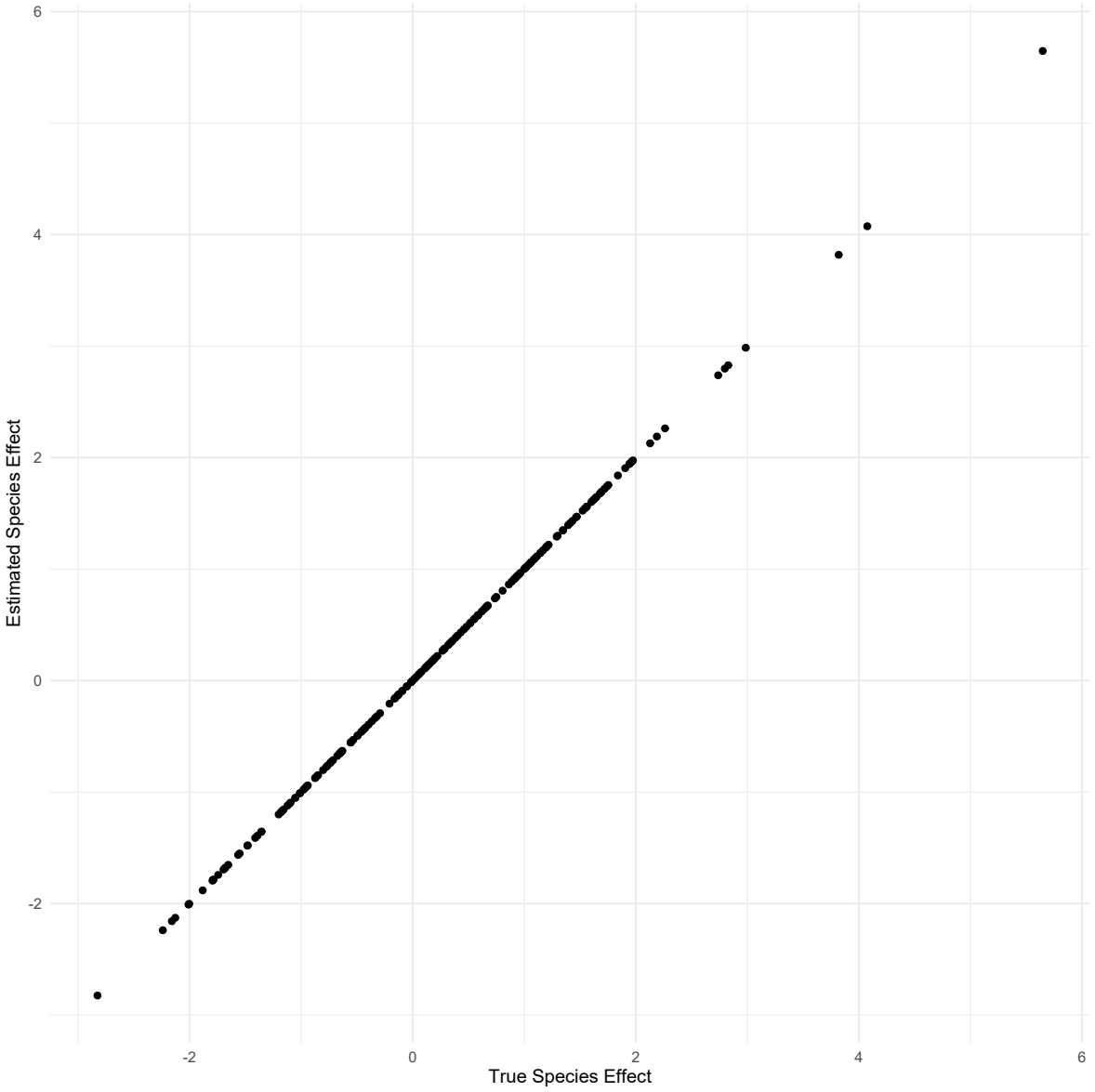
```
##
```

```
## Call:
##    c("inla(formula = resp ~ 0 + intercept + f(spat_id, model = spde), ", "
##    data = inla.stack.data(big_stack, spde = spde_fit), control.compute =
##    list(config = TRUE), ", " control.predictor = list(A =
##    inla.stack.A(big_stack), compute = TRUE, ", " link = 1), num.threads =
##    10)")
## Time used:
##     Pre = 0.375, Running = 25.8, Post = 0.37, Total = 26.5
## Fixed effects:
##            mean    sd 0.025quant 0.5quant 0.975quant  mode kld
## intercept 0.183 0.084      0.017    0.184      0.347 0.184   0
##
## Random effects:
##   Name     Model
##     spat_id SPDE2 model
##
## Model hyperparameters:
##                                           mean       sd 0.025quant 0.5quant
## Precision for the Gaussian observations 18275.33 1.82e+04    1235.33 12879.62
## Range for spat_id                           2.34 2.78e-01       1.84     2.32
## Stdev for spat_id                           2.00 1.00e-01       1.82     2.00
##                                         0.975quant    mode
## Precision for the Gaussian observations   66488.99 3360.46
## Range for spat_id                             2.93    2.29
## Stdev for spat_id                             2.21    1.99
##
## Expected number of effective parameters(stdev): 199.91(0.129)
## Number of equivalent replicates : 1.00
##
## Marginal log-Likelihood:  -254.26
## Posterior marginals for the linear predictor and
##  the fitted values are computed
```

```
sim_fine2$combined_plot
```




```
spat_index <- inla.stack.index(sim_fine2$fit$stack, "loc")
spec_spat_effect <- sim_fine2$fit$mod$summary.fitted.values[spat_index$data, ]

ggplot(sim_fine2$fit$spec_df %>% dplyr::mutate(preds = spec_spat_effect$mean +
                                  sim_fine2$fit$mod$summary.fixed$mean), aes(u, preds)) +
  geom_point() +
  ylab("Estimated Species Effect") +
  xlab("True Species Effect") +
  theme_minimal()
```



The answer is indeed yes! So the emerging picture here is that regardless of how many species or points you have per species, the spatial model will tend to do a good job capturing the true spatial pattern as long as that spatial pattern is not on a scale too much smaller than the species' range sizes. In that case the model will find any larger scale pattern that might happen to exist when aggregating the finer pattern to a larger scale. This will often still model the species-level effect sufficiently. On the other hand, if true spatial structure is on a much finer scale than the species' distribution, it is hard to argue that this spatial structure is relevant to an analysis at the level of the species. It is also worth noting that we have chosen our priors to not put very much weight on range parameters this low, since that would allow the spatial effect to model spatial structure finer than that we see in our environmental predictors. We don't want that because then the spatial effect can absorb most of the variation in the response because it can fit very small-scale fluctuations leading to overfitting. See the manuscript for more discussion of this issue.

Hopefully this set of simulations is enough to convince you that estimating and accounting for spatial structure when analysing species-level responses is possible and reasonably accurate, and worth doing! We certainly strongly believe that this is the best way to approach the problem, when spatial data is available.

Below is the session info for this run of the tutorial, which shows which versions of each package I am using.

```
devtools::session_info()
```

```
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 3.6.2 (2019-12-12)
##  os       Windows 10 x64
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  English_Australia.1252
##  ctype    English_Australia.1252
##  tz       Australia/Sydney
##  date     2020-04-06
##
## - Packages -------------------------------------------------------------------
##  package            * version  date       lib source
##  assertthat           0.2.1    2019-03-21 [1] CRAN (R 3.6.2)
##  backports            1.1.5    2019-10-02 [1] CRAN (R 3.6.1)
##  callr                3.4.3    2020-03-28 [1] CRAN (R 3.6.3)
##  class                7.3-15   2019-01-01 [1] CRAN (R 3.6.2)
##  classInt             0.4-2    2019-10-17 [1] CRAN (R 3.6.2)
##  cli                  2.0.2    2020-02-28 [1] CRAN (R 3.6.3)
##  codetools            0.2-16   2018-12-24 [1] CRAN (R 3.6.2)
##  colorspace           1.4-1    2019-03-18 [1] CRAN (R 3.6.1)
##  crayon               1.3.4    2017-09-16 [1] CRAN (R 3.6.2)
##  crul                 0.9.0    2019-11-06 [1] CRAN (R 3.6.3)
##  curl                 4.3      2019-12-02 [1] CRAN (R 3.6.2)
##  DBI                  1.1.0    2019-12-15 [1] CRAN (R 3.6.2)
##  desc                 1.2.0    2018-05-01 [1] CRAN (R 3.6.2)
##  devtools             2.2.2    2020-02-17 [1] CRAN (R 3.6.2)
##  digest               0.6.25   2020-02-23 [1] CRAN (R 3.6.2)
##  dotCall64          * 1.0-0    2018-07-30 [1] CRAN (R 3.6.3)
##  dplyr              * 0.8.5    2020-03-07 [1] CRAN (R 3.6.3)
##  e1071                1.7-3    2019-11-26 [1] CRAN (R 3.6.2)
##  ellipsis             0.3.0    2019-09-20 [1] CRAN (R 3.6.2)
##  evaluate             0.14     2019-05-28 [1] CRAN (R 3.6.2)
##  extrafont            0.17     2014-12-08 [1] CRAN (R 3.6.2)
##  extrafontdb          1.0      2012-06-11 [1] CRAN (R 3.6.0)
##  fansi                0.4.1    2020-01-08 [1] CRAN (R 3.6.2)
##  farver               2.0.3    2020-01-16 [1] CRAN (R 3.6.2)
##  fields             * 10.3     2020-02-04 [1] CRAN (R 3.6.3)
##  foreach            * 1.4.8    2020-02-09 [1] CRAN (R 3.6.3)
##  foreign              0.8-72   2019-08-02 [1] CRAN (R 3.6.2)
##  fs                   1.3.1    2019-05-06 [1] CRAN (R 3.6.2)
##  gdtools              0.2.2    2020-04-03 [1] CRAN (R 3.6.2)
##  geojson              0.3.2    2019-01-31 [1] CRAN (R 3.6.3)
##  geojsonio            0.9.0    2020-02-13 [1] CRAN (R 3.6.3)
##  geojsonlint          0.4.0    2020-02-13 [1] CRAN (R 3.6.3)
##  ggplot2            * 3.3.0    2020-03-05 [1] CRAN (R 3.6.3)
##  ggrepel              0.8.2    2020-03-08 [1] CRAN (R 3.6.3)
##  glue                 1.3.2    2020-03-12 [1] CRAN (R 3.6.2)
##  gtable               0.3.0    2019-03-25 [1] CRAN (R 3.6.2)
##  hms                  0.5.3    2020-01-08 [1] CRAN (R 3.6.2)
```

```
##  hrbrthemes           0.8.0    2020-03-06 [1] CRAN (R 3.6.3)
##  htmltools            0.4.0    2019-10-04 [1] CRAN (R 3.6.2)
##  httpcode             0.2.0    2016-11-14 [1] CRAN (R 3.6.0)
##  ids                * 1.0.1    2017-05-31 [1] CRAN (R 3.6.3)
##  INLA               * 20.03.17 2020-03-17 [1] local
##  iterators            1.0.12   2019-07-26 [1] CRAN (R 3.6.3)
##  jqr                  1.1.0    2018-10-22 [1] CRAN (R 3.6.3)
##  jsonlite             1.6.1    2020-02-02 [1] CRAN (R 3.6.2)
##  jsonvalidate         1.1.0    2019-06-25 [1] CRAN (R 3.6.3)
##  KernSmooth           2.23-16  2019-10-15 [1] CRAN (R 3.6.2)
##  knitr                1.28     2020-02-06 [1] CRAN (R 3.6.3)
##  labeling             0.3      2014-08-23 [1] CRAN (R 3.6.0)
##  lattice              0.20-38  2018-11-04 [1] CRAN (R 3.6.2)
##  lazyeval             0.2.2    2019-03-15 [1] CRAN (R 3.6.2)
##  lifecycle            0.2.0    2020-03-06 [1] CRAN (R 3.6.3)
##  lwgeom               0.2-1    2020-01-31 [1] CRAN (R 3.6.3)
##  magrittr             1.5      2014-11-22 [1] CRAN (R 3.6.2)
##  maps               * 3.3.0    2018-04-03 [1] CRAN (R 3.6.2)
##  maptools             0.9-9    2019-12-01 [1] CRAN (R 3.6.3)
##  Matrix             * 1.2-18   2019-11-27 [1] CRAN (R 3.6.2)
##  MatrixModels         0.4-1    2015-08-22 [1] CRAN (R 3.6.3)
##  memoise              1.1.0    2017-04-21 [1] CRAN (R 3.6.2)
##  munsell              0.5.0    2018-06-12 [1] CRAN (R 3.6.2)
##  patchwork          * 1.0.0    2019-12-01 [1] CRAN (R 3.6.2)
##  pillar               1.4.3    2019-12-20 [1] CRAN (R 3.6.2)
##  pkgbuild             1.0.6    2019-10-09 [1] CRAN (R 3.6.2)
##  pkgconfig            2.0.3    2019-09-22 [1] CRAN (R 3.6.2)
##  pkgload              1.0.2    2018-10-29 [1] CRAN (R 3.6.2)
##  prettyunits          1.1.1    2020-01-24 [1] CRAN (R 3.6.2)
##  processx             3.4.2    2020-02-09 [1] CRAN (R 3.6.2)
##  ps                   1.3.2    2020-02-13 [1] CRAN (R 3.6.2)
##  purrr                0.3.3    2019-10-18 [1] CRAN (R 3.6.2)
##  R6                   2.4.1    2019-11-12 [1] CRAN (R 3.6.2)
##  raster               3.0-12   2020-01-30 [1] CRAN (R 3.6.2)
##  Rcpp                 1.0.4    2020-03-17 [1] CRAN (R 3.6.3)
##  readr              * 1.3.1    2018-12-21 [1] CRAN (R 3.6.2)
##  remotes              2.1.1    2020-02-15 [1] CRAN (R 3.6.2)
##  rgdal                1.4-8    2019-11-27 [1] CRAN (R 3.6.2)
##  rgeos                0.5-2    2019-10-03 [1] CRAN (R 3.6.3)
##  rlang                0.4.5    2020-03-01 [1] CRAN (R 3.6.3)
##  rmapshaper         * 0.4.3    2020-01-28 [1] CRAN (R 3.6.3)
##  rmarkdown            2.1      2020-01-20 [1] CRAN (R 3.6.3)
##  rnaturalearth      * 0.1.0    2017-03-21 [1] CRAN (R 3.6.3)
##  rnaturalearthhires   0.2.0    2020-04-06 [1] local
##  rprojroot            1.3-2    2018-01-03 [1] CRAN (R 3.6.2)
##  rstudioapi           0.11     2020-02-07 [1] CRAN (R 3.6.2)
##  Rttf2pt1             1.3.8    2020-01-10 [1] CRAN (R 3.6.2)
##  scales               1.1.0    2019-11-18 [1] CRAN (R 3.6.2)
##  scico              * 1.1.0    2018-11-20 [1] CRAN (R 3.6.2)
##  sessioninfo          1.1.1    2018-11-05 [1] CRAN (R 3.6.2)
##  sf                 * 0.8-1    2020-01-28 [1] CRAN (R 3.6.2)
##  sp                 * 1.4-1    2020-02-28 [1] CRAN (R 3.6.3)
##  spam               * 2.5-1    2019-12-12 [1] CRAN (R 3.6.3)
##  splancs              2.01-40  2017-04-16 [1] CRAN (R 3.6.3)
```

```
##  stringi             1.4.6    2020-02-17 [1] CRAN (R 3.6.2)
##  stringr             1.4.0    2019-02-10 [1] CRAN (R 3.6.2)
##  systemfonts         0.1.1    2019-07-01 [1] CRAN (R 3.6.3)
##  testthat            2.3.2    2020-03-02 [1] CRAN (R 3.6.3)
##  tibble              3.0.0    2020-03-30 [1] CRAN (R 3.6.2)
##  tidyselect          1.0.0    2020-01-27 [1] CRAN (R 3.6.2)
##  units               0.6-5    2019-10-08 [1] CRAN (R 3.6.2)
##  usethis             1.5.1    2019-07-04 [1] CRAN (R 3.6.2)
##  utf8                1.1.4    2018-05-24 [1] CRAN (R 3.6.2)
##  uuid                0.1-4    2020-02-26 [1] CRAN (R 3.6.3)
##  V8                  3.0.1    2020-01-22 [1] CRAN (R 3.6.2)
##  vctrs               0.2.4    2020-03-10 [1] CRAN (R 3.6.3)
##  withr               2.1.2    2018-03-15 [1] CRAN (R 3.6.2)
##  xfun                0.12     2020-01-13 [1] CRAN (R 3.6.3)
##  yaml                2.2.1    2020-02-01 [1] CRAN (R 3.6.2)
##
## [1] C:/R/R-3.6.2/library
```