Visualizing speciation rates with IDW interpolation

Weston Testo

03 May 2021

This walkthrough details an approach to visualize spatial distribution of speciation rates via interpolation. Later on, I'll be using two datasets on the Neotropical lycophyte genus *Phlegmariurus*: speciation rate estimates from Testo *et al.* (2019) and ~3500 specimen occurrence records of 90+ species from GBIF. You can access these data and script in this repository. Though we'll be focusing on speciation rates, this approach can be applied to any number of species traits.

Before we get started with the data, here's some context for this work. I was inspired to work on this after reading this really nice paper on orchids by Oscar Pérez Escobar and colleagues in *New Phytologist* in 2017. In that paper, they generated speciation rates from a time-calibrated phylogeny, associated those rates with specimen-based occurrence records, and used inverse-distance weighted (IDW) spatial interpolation in ArcMap to "fill in the gaps" between occurrence points. For a nice explanation of how spatial interpolation works, see this nice post by Manny Gimond.

Here, I am re-envisioned Pérez Escobar *et al.*'s approach with implementation in R. This code mostly relies on the **sf** package for manipulation of spatial data because I like the data structure used by the package and its operations are (to me) intuitive. Of course, you can carry out the same analysis using **sp** based packages, if that is your preference. The interpolation will be done using the **gstat** package.

Why do it this way?

- Implementation in R for improved reproducibility and flexibility
  - Can be run "out of the box" at any spatial extent and re-run easily
- Speed
  - Depending on scale, atleast an order of magnitude faster than QGIS/ArcGIS
- Realistic interpolation extent
  - By making a buffered clipping mask, we avoid unrealistic visualizations

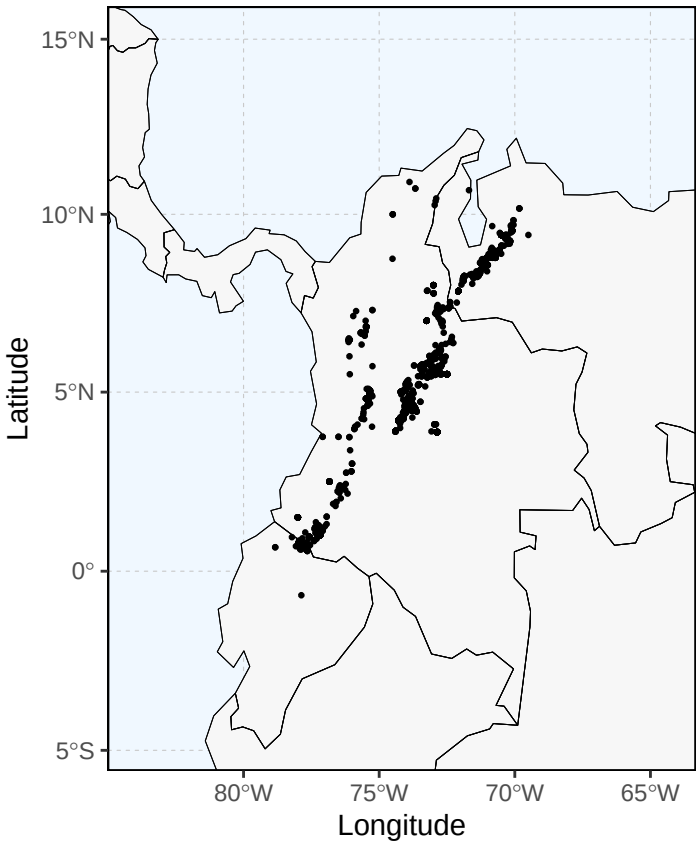
---

Why use a buffer to make a clipping mask?

A slight diversion Because the whole point of interpolation is to estimate values between observed points, it can produce biologically unrealistic results in situations where your study group is not distributed throughout the extent of the interpolation area. This can be overcome by employing a clipping mask, as we'll see.

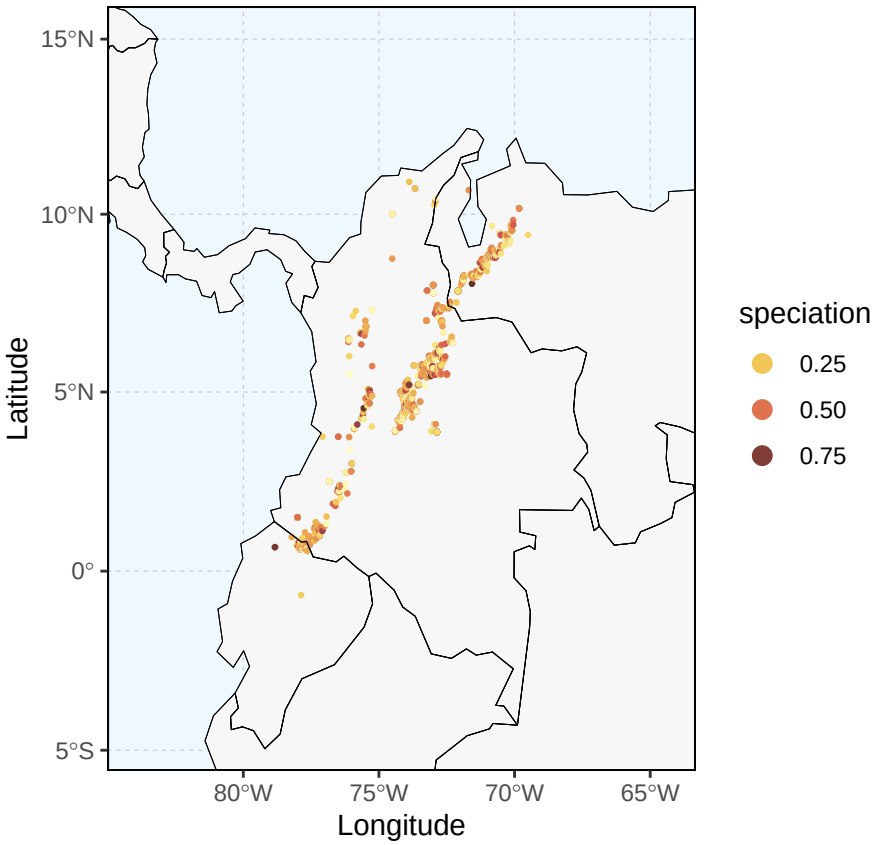
Consider the following example:

The following map shows the distribution of *Espeletia*, a genus in the sunflower family that is only found at high elevations in the northern Andes of South America. **Note**:These data are straight from GBIF and uncleaned, so there are a few errors, like that point in the Chocó of western Colombia.

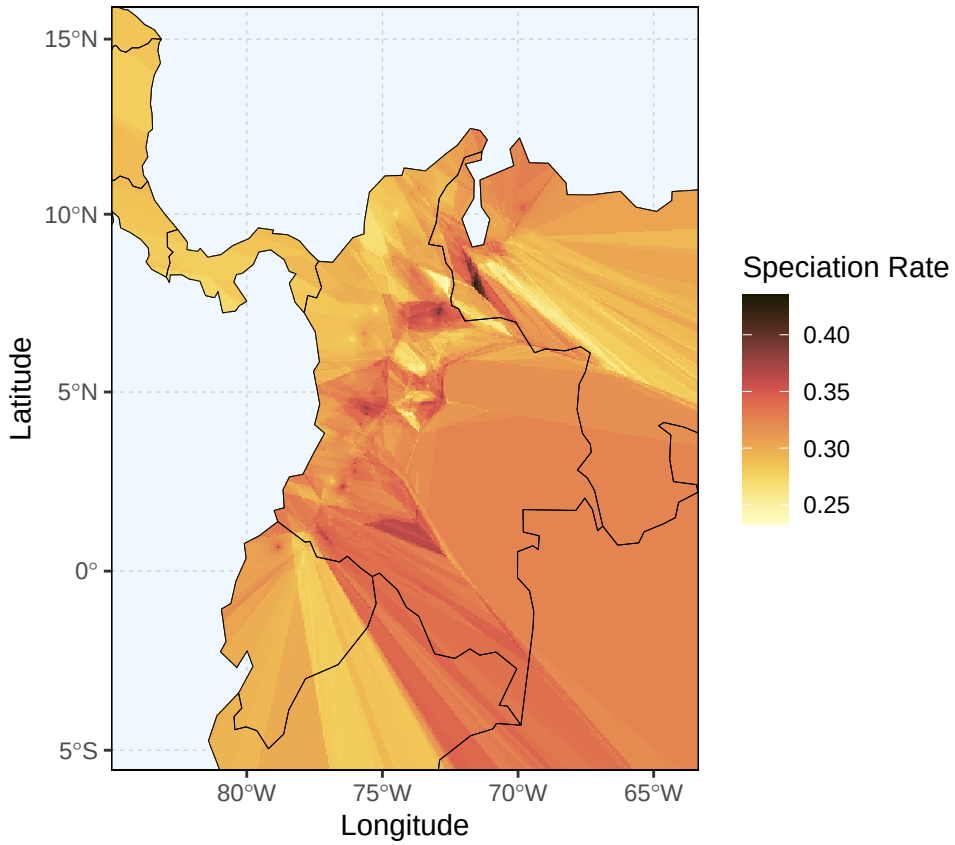


As you can see, the distribution of *Espeletia* is restricted to high elevation habitats in the Andes and associated mountain ranges. Intervening lowland areas (like the Middle Magdalena region in north-central Colombia) are devoid of the genus.

Next, we can see the same map but with the points colored by speciation rate. To keep things simple, I have simply simulated speciation rates with a normal distribution centered on 0.3 events$^{\text{-MY}}$.

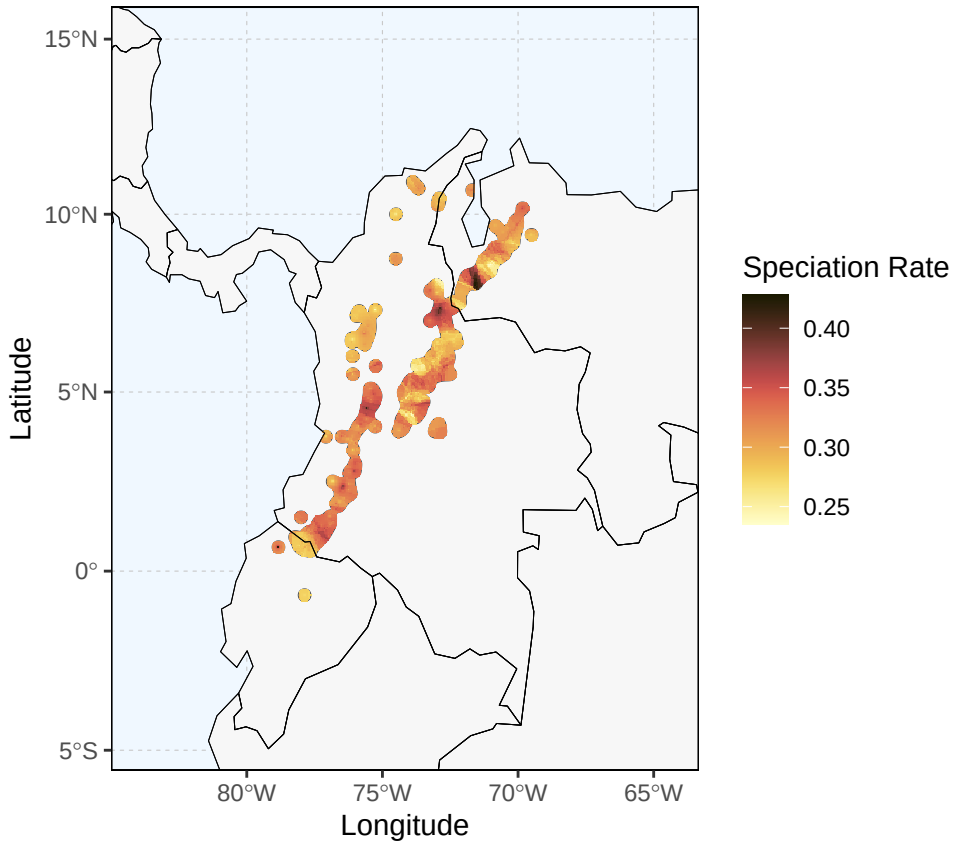


This is interesting, but interpolation of speciation rate estimates between occurrence points would help us to overcome gaps in sampling and potentially make for a more interesting and informative visualization. However, if we carry out interpolation across the current map extent, we get this:



This looks *terrible*, and for good reason – we are interpolating speciation rates across areas where ***Espeletia* doesn't occur**. In addition to looking terrible, this visualization just doesn't make much sense – areas where there are **no** *Espeletia* are shown as having relatively high speciation rates. This is caused by interpolation of rates over areas outside the range of the genus. While this is a rather dramatic case, we can easily imagine lots of scenarios where this could happen.

This is where a clipping mask comes in handy. If we apply a circular buffer of 50 km diameter around each occurrence point and use that to clip the extent of the interpolation, we get a result that makes sense:



Now that we've settled the general approach, let's try this with real data.

Step-by-step walkthrough (with real data)

Loading & checking data First, we'll want to load the packages we'll be using:

```
library(raster)
library(sp)
library(ggplot2)
library(sf)
library(dplyr)
library(rnaturalearth)
library(scico)
library(smoothr)
library(gstat)
```

As I mentioned previously, **sf** and **gstat** will be doing a lot of the heavy lifting here, but several of the other packages help with data manipulation and visualization. In particular, I want to point out Thomas Lin Pedersen's **scico** package, which accesses the *scico* color palettes developed by Fabio Crameri. These are perceptually uniform and colorblind safe color palettes; for more info, see this page.

Next, we'll set a projection for the project (in this case, a Behrmann equal-area projection) and read in a world map using the **rnaturalearth** package.

```
behrmann<-"+proj=cea +lon_0=0 +lat_ts=30 +x_0=0 +y_0=0
+datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"

world_temp<-ne_countries(returnclass = "sf")
world<-st_transform(world_temp, crs = behrmann)
```

Now we'll load the CSV with our occurrence and speciation rate data. The lat/long data are in decimal degrees (WGS84).

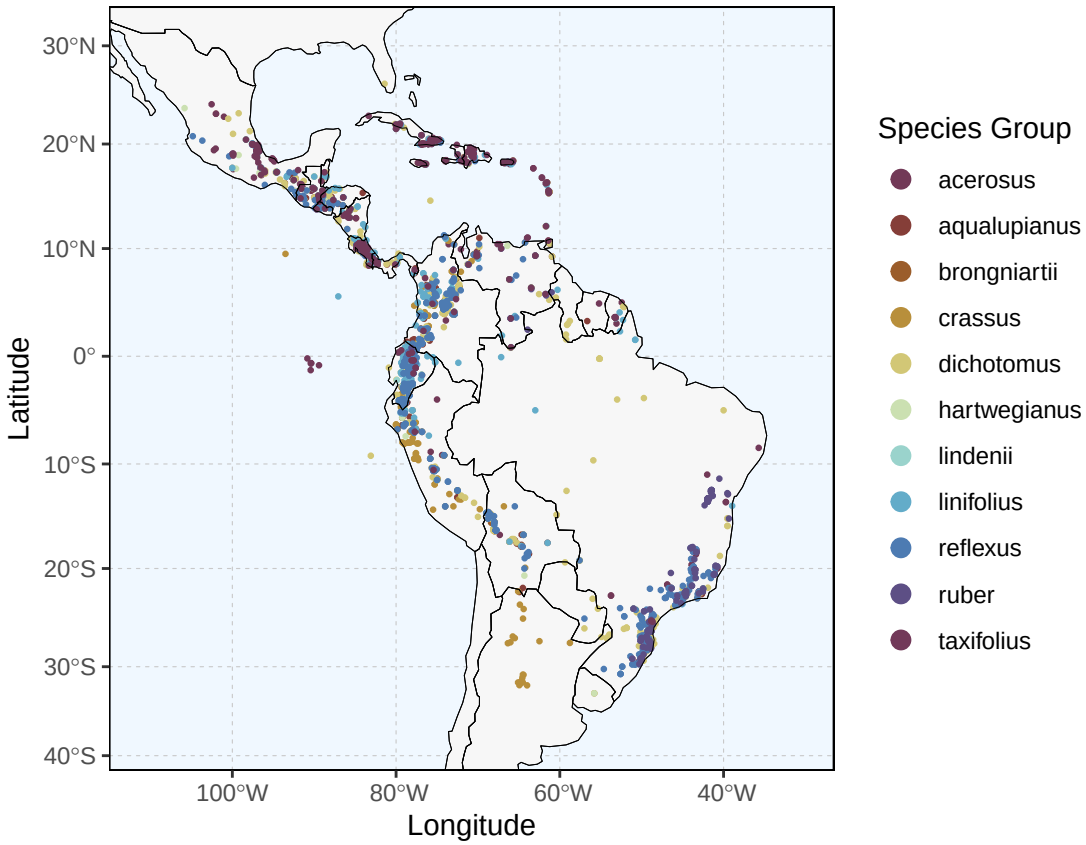
```
points_temp<-read.csv("Phlegmariurus_occ.csv")

head(points_temp)
```

```
##                    Taxon SpeciesGroup decimalLatitude decimalLongitude  Rate
## 1 Phlegmariurus acerosus     acerosus        15.74220        -90.17920 0.142
## 2 Phlegmariurus acerosus     acerosus        14.66666        -88.70000 0.142
## 3 Phlegmariurus acerosus     acerosus        14.53333        -88.66666 0.142
## 4 Phlegmariurus acerosus     acerosus        14.55000        -88.66666 0.142
## 5 Phlegmariurus acerosus     acerosus        15.08330        -88.33330 0.142
## 6 Phlegmariurus acerosus     acerosus        10.30000        -84.75000 0.142
```

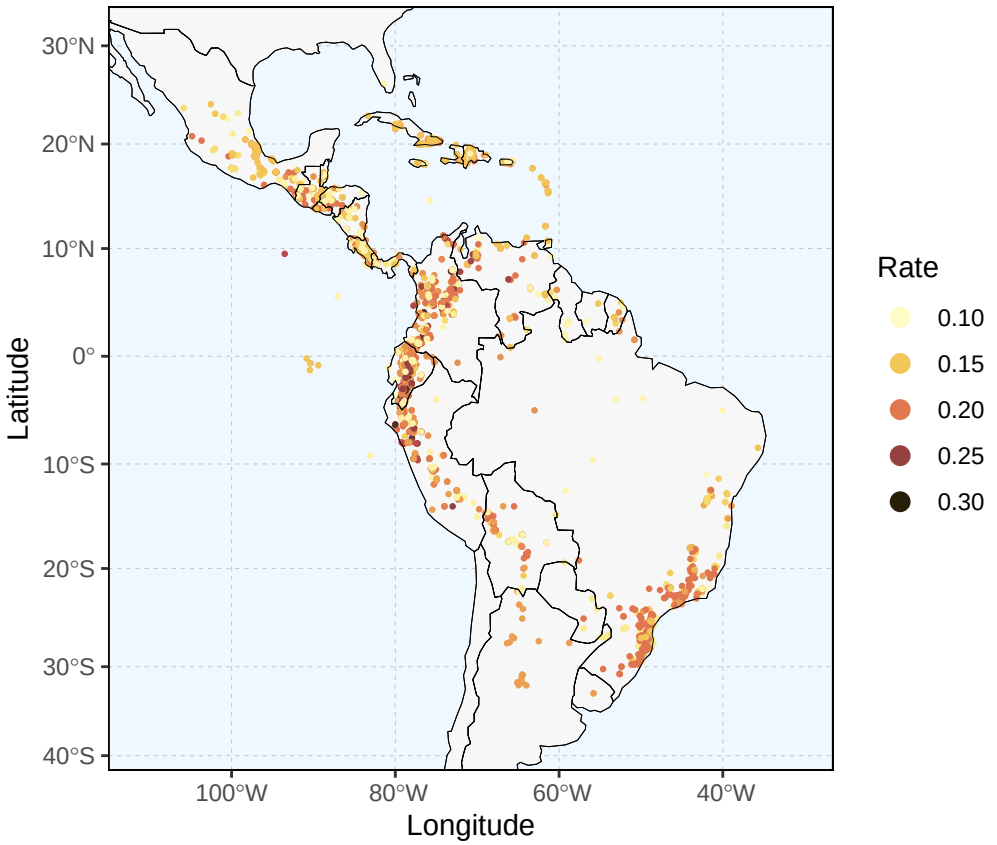
Since we'll be working with the **sf** package, we'll convert our data to a Simple Features object, reproject the coordinates, and create a bounding box that extends 500km beyond our points (we'll use this to define the extent of our plots). Note that in the first line of code we are designating coordinates as – this corresponds to the column numbers of out our longitude and latitude data, respectively (see above).

```
points_temp<-st_as_sf(points_temp, coords = c(4,3),
crs= '+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0')

points<-st_transform(points_temp, crs = behrmann)

limit <- st_buffer(points, dist = 500000) %>% st_bbox()
```

A quick check to make sure our reprojection looks OK –



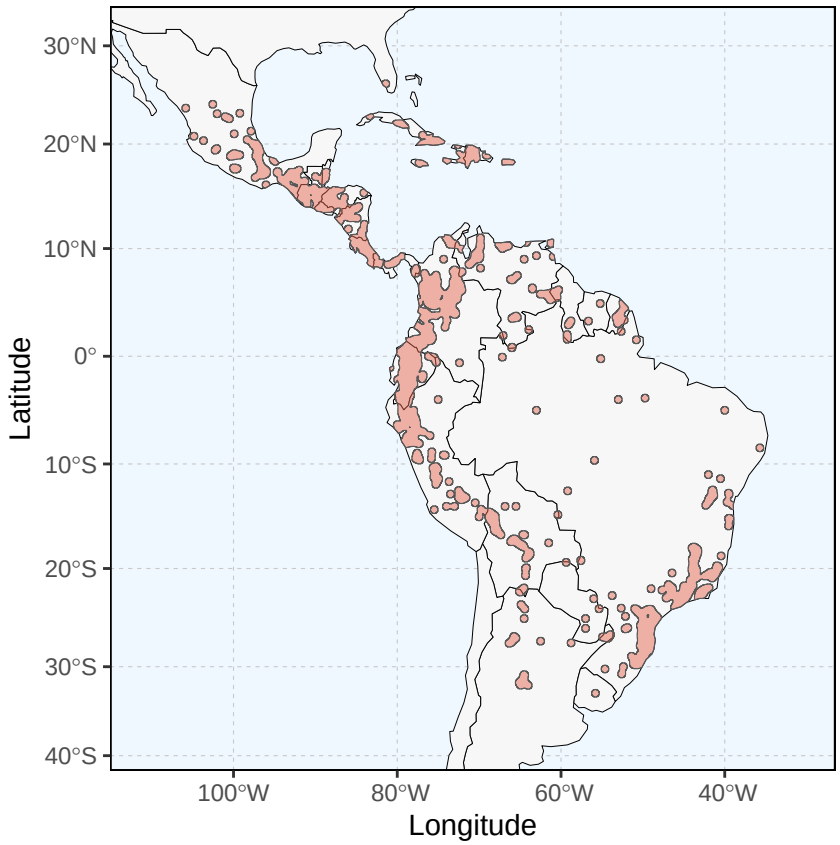
This looks right... *Phlegmariurus* is well-represented in the Andes, southeastern Brazil, and Mesoamerica. Let's take a quick look at the distribution of speciation rates by simply coloring occurrence points by tip rates.



As we can see, the most rapidly diversifying lineages are principally distributed in the northern and central Andes. This makes sense, given the rapid speciation in the *Phlegmariurus crassus* species group in páramo and puna habitats in this region. Now we have a general sense of what to expect from interpolation of these values.

Creating a buffer Let's start by creating our clipping mask that will define the bounds of our interpolation. We'll do this by creating circular buffers of 100km around each occurrence point, fusing overlapping buffers together, clipping any buffer area in the ocean, and smoothing the edges of the buffer. The size of the buffer can be modified to fit the region/taxon you are working with. Note: the 1m buffer is to avoid potential problems with the geometry of the polygon.

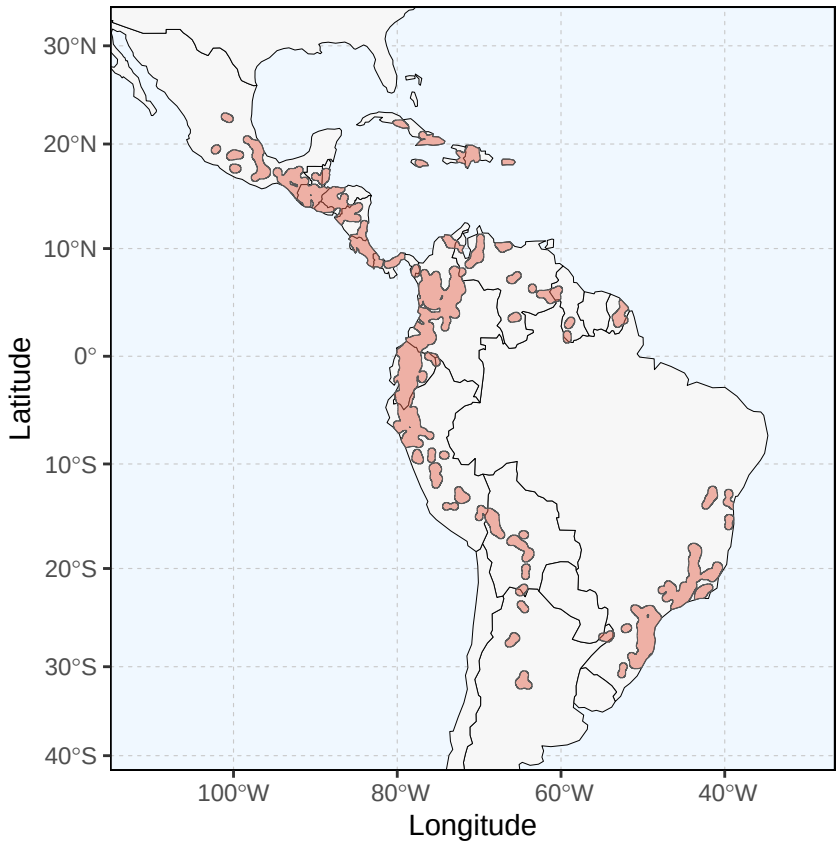
```
buffer<-st_buffer(points$geometry, dist = 50000, nQuadSegs = 50) %>%
  st_union()%>%
  smooth(method="ksmooth", smoothness=40) %>%
  st_intersection(world) %>%
  st_buffer(dist=1) %>%
  st_union()
```



This looks pretty good overall. Most of the range has been fused into a small number of contiguous polygons, and there are a few outlying circular buffers. If we wanted to keep just the "core range" and drop the polygons defined by single points, we can do some pretty simple housekeeping based on that fact that we can figure out the approximate size of the isolated polygons using some basic geometry: $A = \pi * r^2 \ldots$

```
A = 3.14*(50000^2) ## approx. area of single-point buffers with 50km radius

bufferManyParts<-buffer %>% st_cast("POLYGON") ## make multiple polygons

bufferLarge <- bufferManyParts[as.numeric(st_area(bufferManyParts)) > A]

bufferLarge <- bufferLarge %>% st_cast("MULTIPOLYGON")
```

The result is rather similar to the above plot, but perhaps a bit "cleaner".



For the purposes of this walkthrough, we'll use the untrimmed buffer from here on out, as we want to retain as much information about the clade's distribution as possible. Our next step is to rasterize the buffer so we can easily use it later to clip the interpolation output.

```
buffer_sp <- as(buffer, "Spatial")
```

Creating a grid for interpolation Because the output of IDW interpolation is a raster, we need to make a grid to form the basis for that raster. Each grid cell will correspond to a pixel in the raster, so our choice of cell size will define the resolution of the final raster. Note that increasing the number of grid cells will mean that there will be more "blank" cells to estimate values for, and the resulting raster file will be larger. Figuring out the right balance between resolution and runtime/file size might take some experimentation.

Here, we will use a cell size of 5 x 5km. Because the size grid is defined by the extents of the buffer, the dimensions of the grid will vary depending on your study area. The **Z** field added to the grid at the end will hold the interpolated speciation rates. After the grid is created, we'll rasterize it.

```
grid <- buffer %>%
  st_bbox() %>%  ## create bounding box for limit of grid
  st_as_sfc() %>%
  st_make_grid(cellsize = c(5000, 5000), ## cell size (in meters)
               what = "centers") %>%
  st_as_sf() %>%
  cbind(., st_coordinates(.)) %>%
```

```
  st_drop_geometry() %>%
  mutate(Z = 0) ## additional field for holding rate values

rasterGrid <- grid %>%
  raster::rasterFromXYZ(
    crs = behrmann)
```

Interpolating rates Now that we have created a grid, we can carry out the interpolation. First, we need to develop interpolation model, which we will do in the **gstats** package. This is relatively simple, and involves four parts:

1. A formula (in our case, just the variable of interest: **Rate** ~ 1)
2. The data source (in our case, the object **points**)
3. The min (**nmin**) and max (**nmax**) number of points to use for interpolation of a given cell
4. Inverse distance power (**idp**), which specifies the strength of the local effect. **idp = 0** means there is no effect of distance, higher values (e.g., **idp = 3** indicate a strong effect).

```
IDW_model <- gstat::gstat(
  formula = Rate ~ 1,   ## change name as needed based on your data
  data = as(points, "Spatial"),
  nmax = 50, nmin = 10,
  set = list(idp = 0.5))
```

It is worth experimenting with **idp** and **nmin**/**nmax** is good idea to get a sense of how they affect your results. It is possible to validate the model using LOOCV or other methods, but that is beyond the scope of this walkthrough.

Now that we have defined the model, we can run the interpolation:

```
IDW_output <- interpolate(rasterGrid, IDW_model)
```

We can use the buffer to clip the extent of the interpolation and convert the resulting raster to a tibble so we can visualize it.

```
IDW_output_trimmed<-mask(IDW_output, buffer_sp)

IDW_df <- rasterToPoints(IDW_output_trimmed) %>% as_tibble()
colnames(IDW_df) <- c("X", "Y", "Z")
```

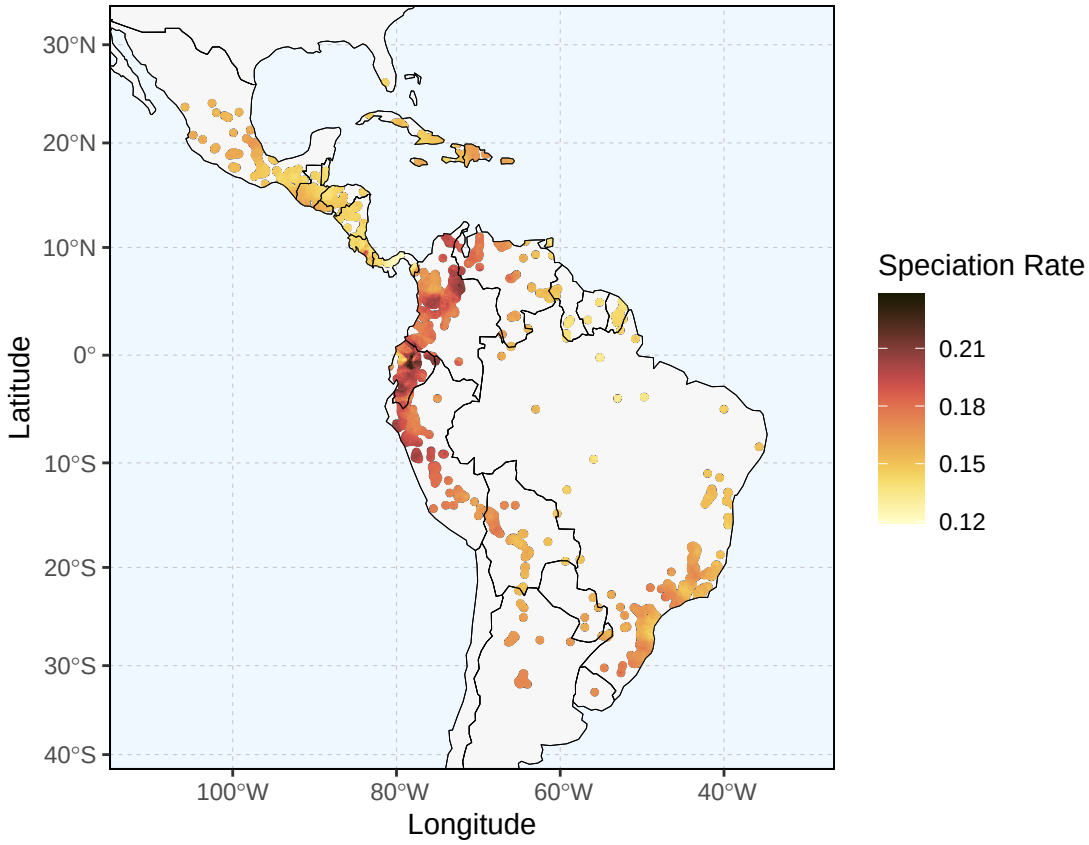
Now we have a tibble with three columns: X, Y, Z. The first two columns represent the coordinates of each cell, and the Z column shows the associated speciation rate. This can easily be be visualized in **ggplot2** using the **geom_tile** aesthetic. For this dataset, the tibble is large – more than 110k rows of data – let's look at a random subset of cells:

```
sample_n(IDW_df,15)
```

```
## # A tibble: 15 x 3
##            X         Y     Z
```

```
##        <dbl>     <dbl> <dbl>
##  1 -6894170.  1060653. 0.179
##  2 -4164170. -2774347. 0.156
##  3 -4259170. -2839347. 0.152
##  4 -5799170. -1884347. 0.149
##  5 -7564170.  -949347. 0.188
##  6 -5029170.   585653. 0.135
##  7 -7664170.  -224347. 0.190
##  8 -8064170.  1570653. 0.147
##  9 -9849170.  2490653. 0.155
## 10 -7199170.   520653. 0.193
## 11 -7064170.  1325653. 0.192
## 12 -6749170.  2355653. 0.165
## 13 -4574170. -2849347. 0.176
## 14 -7334170.   860653. 0.166
## 15 -6294170. -3349347. 0.166
```

Finally, we can visualize the output:



Things look as we expected – high rates in the tropical Andes, much like we saw in the point-based figure previously. But, this gives a much more complete story and looks pretty nice. Along with being an informative figure in itself, there are a lot of data here than can be extracted for downstream analyses.

Enjoy! If you have any questions or suggestions, open an issue or email me a westontesto at gmail.com.