phylin with resistance

Pedro Tarroso
Guillermo Velo-Antón
Sílvia B. Carvalho

December 12, 2019

1 Preamble

In this tutorial we will cover the interpolation of lineage probability with resistance calculation. Although the tutorial focus in resitance distances, as you will see, the method is not limited to this distances. You can use the most appropriate distance metric explaining the spatial relations. The example data is borrowed from Tarroso et al. (2018) and the tutorial will cover partially the methods described there. We want to mantain a simple guide that is easy to follow and dedicated to **phylin** but some extra packages will be needed to follow the guide. Please, make sure you have the package **gdistance**, needed for resistance distance calculations, installed in your R environment.

To install **phylin** please check the *step-by-step tutorial* vignette. You will find instructions on how to use **phylin** including its basic usage, variogram building, model fitting and spatial interpolation of lineage probability.

2 Example data

In this tutorial we will use a simulated environment and genetic distances that are available with the **phylin** package under the name `simulations`.

The example data includes three data sets:

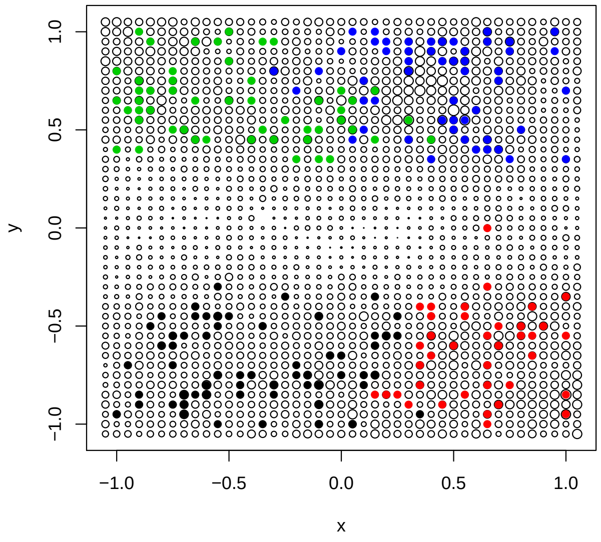
- *simul.env* – A data frame with the coordinates of the grid centroids (1849 cells) and the values of two simulated environmental surfaces for each grid cell.
- *simul.sample* – A table of 200 samples randomly chosen from the simulated presence, with coordinates and lineage classification.
- *simul.gen.dist* – A genetic distance matrix for the 200 samples.

See the original publication (Tarroso et al., 2018) for details on how the simulations were built. The data can be attached to your R session:

```
> library(phylin)
> data(simulations)
```

The grid with the first environmental surface values found in *simul.env* and the samples with different colors per lineage can be plotted with:

```
> size <- (simul.env$env.sur - min(simul.env$env.sur)) / 10
> grid.full <- simul.env[,1:2]
> plot(grid.full, cex = size)
> points(simul.sample[,1:2], col=simul.sample$lineage, pch=16)
```



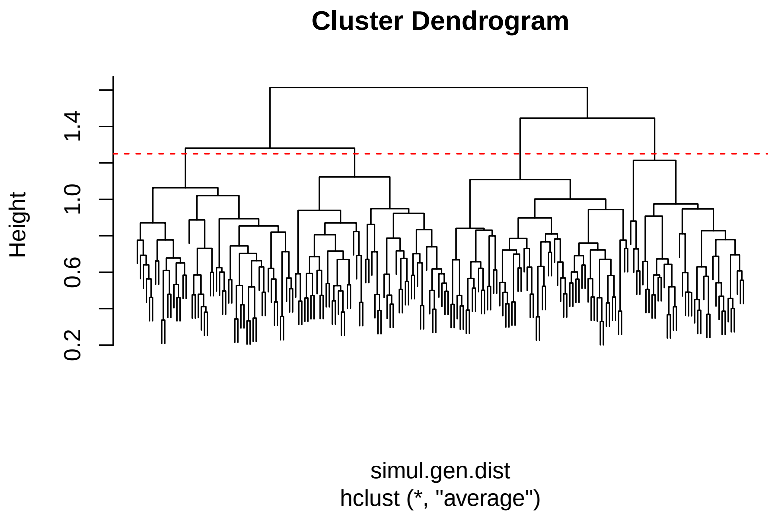
Notice that there are no samples in the edges. Later, when calculating resistance distances, we need to compute a transition matrix. In order to avoid edge effects in the calculation, the grid has an extra line of cells in the edges[1]. The final interpolation will only occur in the smaller grid without the edges. In this step we will remove the edge pixels and prepare a grid for the later interpolations. In this case, the grid with edges is within the interval -1.05 to 1.05, in both axes. So, to remove those cells we have only to retain coordinates X and Y that are smaller than 1.05 in absolute values.

```
> grid <- grid.full[abs(grid.full$x) < 1.05 & abs(grid.full$y) < 1.05,]
```

The simulated genetic distance can be used to build a tree with UPGMA method to show how are the four lineages organized:

```
> plot(hclust(simul.gen.dist, method="average"), labels=FALSE)
> abline(h = 1.25, col='red', lty=2)
```

[1]With real data, the study area is usually larger than the extent of the sample points, and, thus, this is not a common problem. However, being sure that you have a larger area is important in resistance distance calculation, not only to avoid edge effects in the calculation but to increase the probability of finding lower resistance paths that circumvent high resistance areas.



3 Distance calculation

In this section we will build two distance matrices. One is the typical geographical distance matrix based on euclidean distances between coordinates. The other is a resistance distance matrix based on the friction that landscape offers to the movement of individuals.

3.1 Geographical distances

This matrix is generated from the coordinates of the samples:

```
> geo.dist <- dist(simul.sample[,1:2])
```

The `dist` function from R will calculate the euclidean distances based on the coordinates of the samples. On real data sets you should be careful about the projection system used. These distances are not true geographical distances when using real data as they do not take into consideration the curvature of the Earth. For small study areas, the curvature might be negligible and the euclidean distance are a good approximation of the geographical distances.[2].

3.2 Resistance distances

This matrix is more difficult to calculate as it needs other package and multiple steps. In this tutorial we are using resistance distance based on **gdistance** package. You can use any other type of distance (e.g. least-cost path) or a package/software of your choice. For instance, resistance distances were popularized in the landscape genetics community by the CircuitScape software (Shah and McRae, 2008) and it is possible to use within R environment (e.g Peterman, 2018). Although you can use external software, it is much easier if it is integrated with R, particularly for the later interpolation process.

[2]For example, check package **geosphere** for distance calculations on a spheroid.

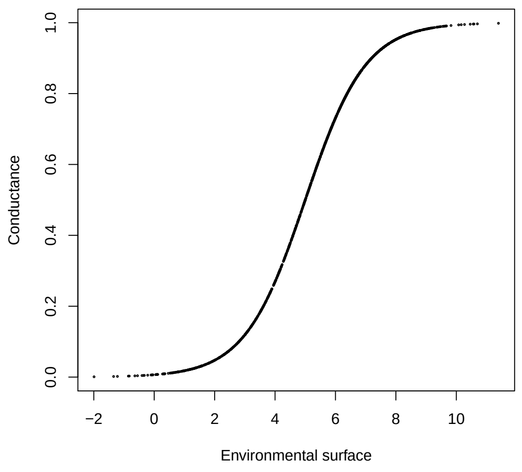
The first step is to generate a conductance surface that will describe the permeability of the landscape to the species movement. Although we will generally use the term resistance distance, we will be using conductance to derive those distances. In fact, conductance is the reciprocal of resistance: $R = \frac{1}{C}$ where R is resistance and C is conductance.

The conductance we will be creating is based on the first environmental surface in the *simul.env* data. Similarly to Tarroso et al. (2018), we apply a logistic function to the data to convert the environmental values to a conductance value in the range from zero to one with the formula

$$C = \frac{k}{1 + e^{-b(x-m)}}$$

where x is the value of the first environmental surface, k is the upper asymptote set to 1, b is the curvature parameter set to 1, and m is the inflection point set to 5. This translates the environmental surface values to conductance on the range $[0, 1]$. See Tarroso et al. (2018) for more details on the methods and on finding the correct parameters with real data.

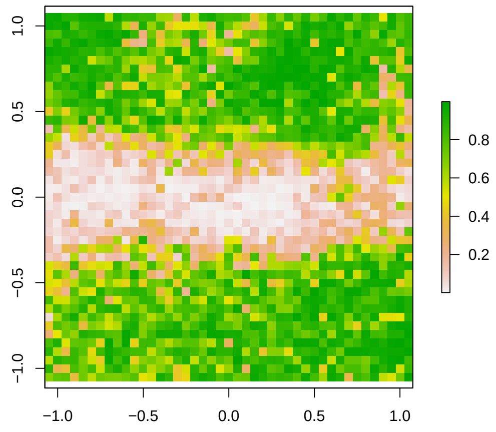
```
> conductance <- 1/(1+exp(-1*(simul.env$env.sur-5)))
> plot(simul.env$env.sur, conductance, cex=0.25,
+      xlab="Environmental surface", ylab = "Conductance")
```



This virtual species disperses more easily through higher values of the environmental surface (higher conductance / lower resistance).

To calculate the resistance matrix we will need to convert the conductance to raster. We will use a function from the **raster** package. However, as **gdistance** depends on **raster**, this package is already loaded in our R session.

```
> library(gdistance)
> conductance <- rasterFromXYZ(data.frame(grid.full, conductance))
> plot(conductance)
```



The transition matrix represents the connectivity between grid locations. The matrix is calculated based on the conductance raster using 8 neighbors, assuming that each pixel is connected to all its nearest orthogonal and diagonal nearest neighbors. The transition matrix has to be geo-corrected in order to correctly represent the connectivity. See van Etten (2017) for more details on the generation of resistance distances with the **gdistance** package.

```
> tr <- transition(conductance, mean, 8)
> tr <- geoCorrection(tr, type="r")
```

With the transition matrix we can finally calculate the resistance distances between the samples:

```
> res.dist <- commuteDistance(tr, as.matrix(simul.sample[,1:2]))
```

The commute distances are equivalent to resistance distance from circuit theory (van Etten, 2017).

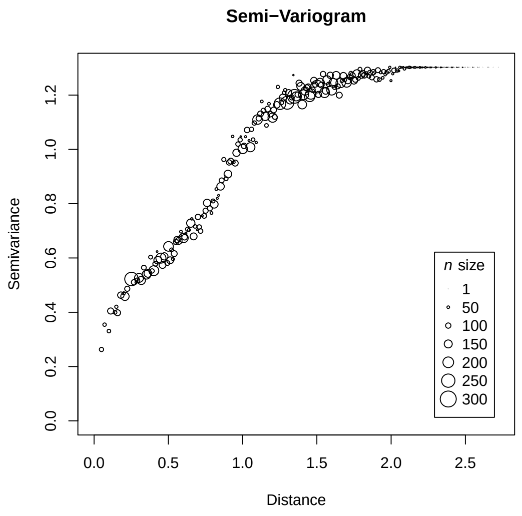
Now we have two distance matrices, the `geo.dist` for geographical distances and `res.dist` for resistance-based distances. These two matrices will be used in the interpolation in order to build the lineages' occurrence probability maps.

4 Building the variograms

Building a variogram and fitting a theoretical model to it is a necessary step for mapping the lineage occurrence probability. We will build two variograms: one for the geographical distances and the other for the resistance-based distances. Our genetic distances are in the matrix `simul.gen.dist`.

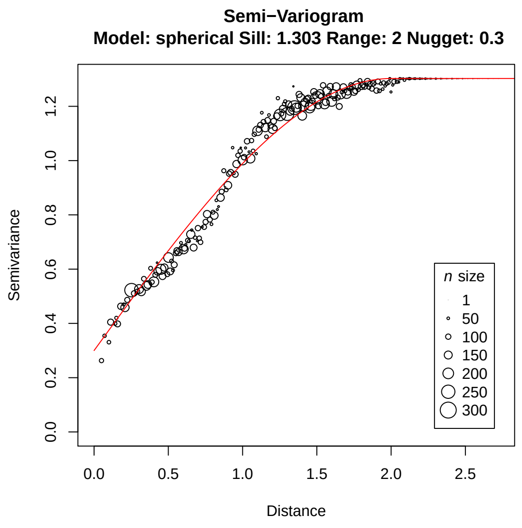
For the geographical distances we use a value of 0.01 for the distance class interval (lag).

```
> gv.geo <- gen.variogram(geo.dist, simul.gen.dist, lag=0.01)
> plot(gv.geo)
```



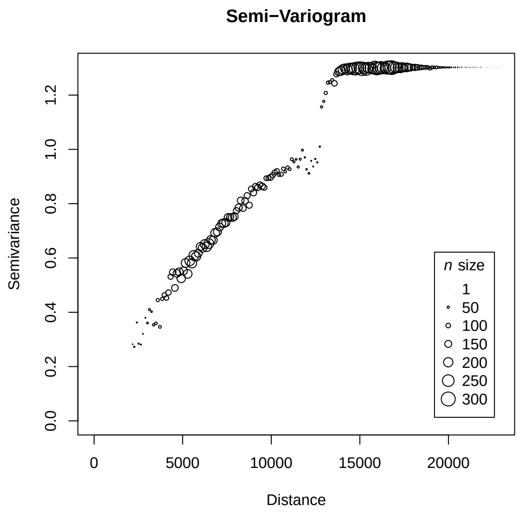
A model with a range value of 2 and a small nugget of 0.3 seems to fit the empirical variogram. We can add it with the following commands:

```
> gv.geo <- gv.model(gv.geo, range=2, nugget=0.3)
> plot(gv.geo)
```



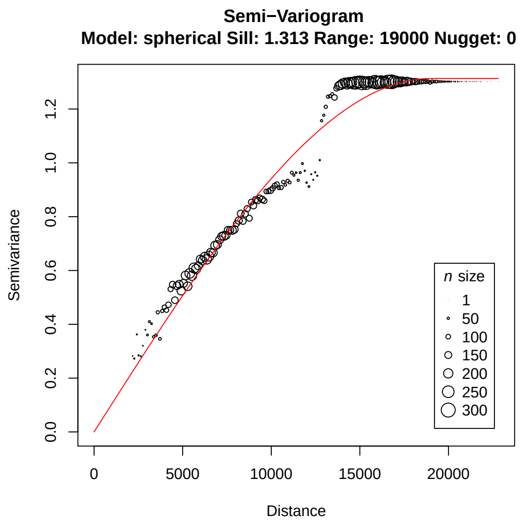
The resistance distance are used to build the second variogram. Since resistance distances are of much larger magnitude, the lag value is also larger.

```
> gv.res <- gen.variogram(res.dist, simul.gen.dist, lag=120)
> plot(gv.res)
```



We can also fit a model to the variogram after visual inspection. This time we assume a zero nugget and a range value of 190.

```
> gv.res <- gv.model(gv.res, range=19000)
> plot(gv.res)
```



The variograms have very different shapes. The resistance distances separate well the lineages below (1 and 2) and above the barrier (3 and 4), resulting in a double inflection point. The model we have fitted to the variogram does not take this into account for simplicity. The simulated environment has a barrier that crosses the study area longitudinally and genetic distances are derived directly from the simulated environment. This forces lineages to be well differentiated on each side of the barriers and to have a genetic divergence that has an high spatial structure. Lineages above and below the barrier are well differentiated genetically and resistance distances are promoting the division by forcing longer distances across the barrier. Processes as species dispersal, permeability of the barrier and allowed gene flow between individuals at both sides of the barrier would smooth out this effect.

We could also assume two different spatial processes occurring at both side of the barrier. With this assumption we could build a variogram for each spatial process (see a similar case in Tarroso et al., 2015). In this case we would have a variogram including lineages 1/2 and another for lineages 3/4 that would be used later in the spatial interpolation of each lineage. This division would result in simpler empirical variograms to which fitting a model would be easier.

For simplicity and because the simulations did not intend to model two fully isolated species, we proceed with a single variogram for resistance distances.

5 Distance functions

The objective of this tutorial is to create a raster representing the probability of lineage occurrence. For this we have to create a simple function that allows to calculate the appropriate distances (geographical or resistance) for each interpolation. The `krig` function needs an external function to calculate distances needed in the interpolation computation. As default, the function will assume simple euclidean distances and no external function is required. However, when using other distances, we have to provide a function that calculates the appropriate distance between a origin and a destination point.

To illustrate better the interpolation process in this tutorial, we will build also a simple function to calculate the euclidean distances[3]. The function has two mandatory arguments, 'from' and 'to', representing the origin and destination points and returns a distance matrix with 'from' locations in rows and 'to' locations in columns. We are going to use the `dist` function from R to do most of the calculations for us. However we still have to wrap it in a function accepting a 'from' and 'to' arguments. We are going to use a small trick: we build a full distance matrix including all points and extract just the portion we need (the 'from' rows and 'to' columns)[4]. The tables 'from' and 'to' must have only two columns representing X and Y coordinates. The function follows a simple structure:

1. Get the number of 'from' locations.
2. Merge 'from' and 'to' locations to a single matrix with X and Y columns.
3. Compute euclidean distances with `dist` between merged locations.
4. Subset the matrix to get the rows that are 'from' locations and the columns that are 'to' locations.
5. Return the distance matrix.

[3]Note again that this function is not needed as, by default, `krig` will calculate euclidean distances.

[4]This is not efficient at all because it performs unneeded computations but it is simple.

Written in R code:

```
> my.geo.dist <- function (from, to) {
+     nf <- nrow(from)
+     allcoords <- rbind(from, to)
+     dist <- as.matrix(dist(allcoords))
+     geo.dist <- dist[1:nf, (nf+1):ncol(dist)]
+     return(geo.dist)
+ }
```

We can try our function with some coordinates. We are using the 3 first samples and the 6 first grid coordinates to calculate distance with our `my.geo.dist` function:

```
> # The 'x' and 'y' columns of the first 3 samples
> sp <- simul.sample[1:3, 1:2]
> # The first 6 locations in the grid
> grd <- grid.full[1:6,]
> # Calculate distances from samples to grid
> my.geo.dist(sp, grd)
```

```
              1        2        3        4        5        6
px274  1.443087 1.394633 1.346291 1.298075 1.250000 1.202082
px1274 1.607016 1.603122 1.600781 1.600000 1.600781 1.603122
px1629 2.500000 2.470324 2.441311 2.412986 2.385372 2.358495
```

Our function seems to be working correctly. We can now proceed to create the function that computes resistance distances. The function is slightly more complex as it needs the 'from' and 'to' arguments but also a transition matrix as a 'tr' argument. The first two arguments are mandatory for the `krig` function and other arguments are optional, depending only on the chosen distance algorithm. As we seen above, the `commuteDistance` function from **gdistance** needs a transition matrix and our function must supply it. We will use the same function structure as before with few modifications:

```
> my.res.dist <- function (from, to, tr) {
+     nf <- nrow(from)
+     allcoords <- as.matrix(rbind(from, to))
+     dist <- as.matrix(commuteDistance(tr, allcoords))
+     my.dist <- dist[1:nf, (nf+1):ncol(dist)]
+     return(my.dist)
+ }
```

We can check the output of the created function with the same coordinates as before:

```
> # Calculate distances from samples to grid
> my.res.dist(sp, grd, tr)
```

```
              1        2        3        4        5        6
px274  16103.50 13906.22 13093.21 13021.95 11886.00 11653.83
px1274 26032.10 23842.33 23047.16 23004.95 21906.43 21721.30
px1629 24528.51 22336.91 21537.42 21488.20 20380.71 20184.39
```

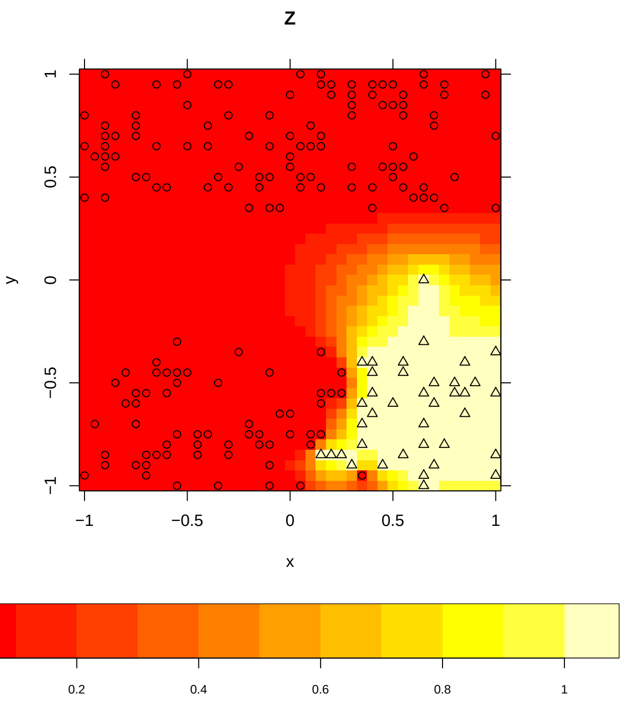
Notice again that we must give the transition matrix to the function.

Now we have the functions for the interpolation[5]. Allowing user defined functions in the interpolation process brings much flexibility to the method. The interpolation is not limited to euclidean distances neither to resistance distances. The user can decide which type of distance better suits his system and apply it.

6 Spatial interpolation

We can proceed with the interpolation of lineage occurrence probability. We convert the sample lineage data into a vector of 0s and 1s representing the lineage presence. For the following example, we will be using the lineage 2. The 'neg.weights' set to *FALSE* allows only positive weights on the computation of the interpolations, resulting in a interpolation within the original range between zero and one.

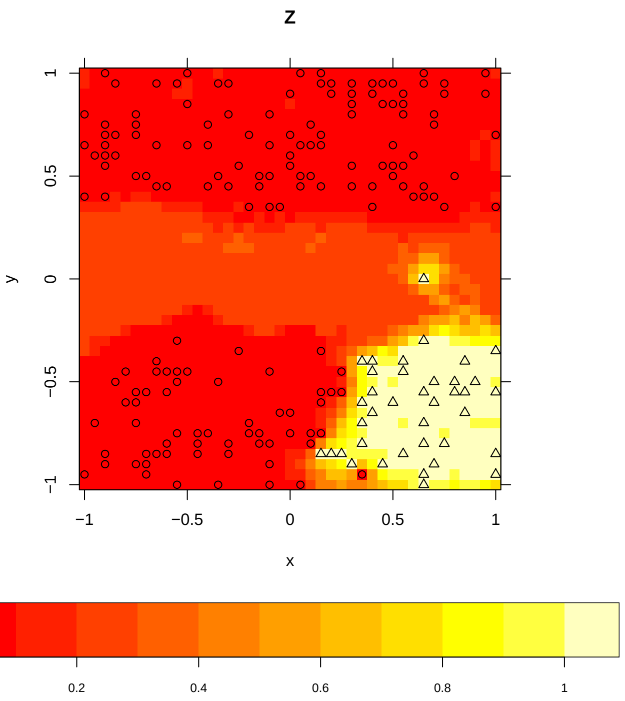
```
> lin <- as.integer(simul.sample$lineage == 2)
> intpl <- krig(lin, simul.sample[,1:2], grid, gv.geo, my.geo.dist,
+                neg.weights=FALSE, verbose=FALSE)
> grid.image(intpl, grid)
> points(simul.sample[,1:2], pch=lin+1)
```



[5]Writing functions might be difficult, particularly for someone without a previous programming experience. There are multiple books, resources and tutorials covering the subject. For example: https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

For resistance distances, the process is the similar. We have to use the function we created before. This function has three arguments: the two mandatory for `krig` that are the 'from' and 'to' and the transition matrix 'tr' argument. We have to pass the transition matrix to our function trough the `krig` function.

```
> lin <- as.integer(simul.sample$lineage == 2)
> intpl <- krig(lin, simul.sample[,1:2], grid, gv.res, my.res.dist, tr=tr,
+                 neg.weights=FALSE, verbose=FALSE)
> grid.image(intpl, grid)
> points(simul.sample[,1:2], pch=lin+1)
```



The barrier effect is neglected with the interpolation considering only geographical distances, resulting in a distribution of the lineage that expands over the area with high resistance. The only factor shaping the distribution of the probability of occurrence is the relative position of the samples. However, by definition, the barrier does not have any presence due to the lack of suitability, and lineages are predicted to expand to the nearest areas to the sampling distribution. On the other hand, the interpolation with resistance distance limits the distribution of the lineage with the limit of the barrier. The barrier has an intermediate value of lineage occurrence due to the lack of samples: all lineages have the same low probability of being present there. Nevertheless, the lineage distribution area is well differentiated with the interpolation values and it detects a potential passage through the barrier.

The process can be automated to produce an interpolation for each lineage with a loop. We first create an empty matrix that will hold all the values and than attribute to each column the results of each lineage interpolation. In the following example we use the same loop for geographical and resistance distances.

```
> geo <- matrix(NA, nrow(grid), 4)
> res <- matrix(NA, nrow(grid), 4)
> for (l in 1:4) {
+     lin <- as.integer(simul.sample$lineage == l)
+     geo[,l] <- krig(lin, simul.sample[,1:2], grid, gv.geo, my.geo.dist,
+                   neg.weights=FALSE, verbose=FALSE)$Z
+     res[,l] <- krig(lin, simul.sample[,1:2], grid, gv.res, my.res.dist, tr=tr,
+                   neg.weights=FALSE, verbose=FALSE)$Z
+ }
```

You can view the results using the `grid.image` function as before. You can also easily convert to a raster object using the **raster** package:

```
> geo.raster <- rasterFromXYZ(data.frame(grid, geo))
> res.raster <- rasterFromXYZ(data.frame(grid, res))
```

You can write the raster using the `writeRaster` function from the **raster** package. If you save the raster created it will result in a multi-band raster where each band is a lineage occurrence.

References

Peterman, W. E. (2018). ResistanceGA: An R package for the optimization of resistance surfaces using genetic algorithms. *Methods in Ecology and Evolution*, (00):1–10.

Shah, V. and McRae, B. (2008). Circuitscape: a tool for landscape ecology. *Proceedings of the 7th Python in Science Conference*, pages 62–65.

Tarroso, P., Carvalho, S. B., and Velo-Antón, G. (2018). Phylin v2: improving the phylogenetic lineage interpolation method including uncertainty and user-defined distance metrics. Submitted.

Tarroso, P., Velo-Antón, G., and Carvalho, S. B. (2015). Phylin: an R Package for Phylogeographic Interpolation. *Molecular Ecology Resources*, 15:349–357.

van Etten, J. (2017). R Package gdistance: Distances and Routes on Geographical Grids. *Journal of Statistical Software*, 76(13):21.