Landscape geneticis - Katoomba

Niko Balkenhol & Bernd Gruber

2019-08-01

Contents


UNIVERSITY OF
CANBERRA
INSTITUTE FOR
APPLIED ECOLOGY
Centre for
Biodiversity
Analysis
GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN
MACQUARIE
University

Forword

The idea of this tutorial is:

- You run it on your on pace.
- During the tutorial there will be tasks to be solved (look out for the penguins)
- If you get stuck, please ask or refer to the hint/solutions using the `hint()` and `solution()`functions.
- Please ask and discuss as much as you like the content.
- The tutorial runs for 3 hours, so please be aware that not all the content can be covered in detail. Again please discuss and ask us as much as you like regarding the topics.

Have fun,
Niko & Bernd

Introduction

The tutorial aims to teach you how to run a landscape genetic analysis. This requires from you two things. An understanding of genetics and how we meassure population structure (either on a (sub)population level or on an individual level). As we try to be "modern", we will use SNP data, so some people would call the tutorial 'landscape genomics'. The second requirement is an understanding of spatial data (point and raster data), as we need a representation of individuals in space to do landscape genomics. The main issues here are the ideas of coordinate systems, projection and a landscape represented by resistance values.

- The tutorial consists of three parts:

1. Load your data [genetics and landscape] into R
2. Resistance surfaces
3. Analyse the distance matrices [(partial) mantel tests, MMRR, gdm, MCMC (sunder)]
4. Other bits
5. Final exam (just joking)

0. Install packages and preparation

For this R session we need a number of packages to be installed. The good news is that all packages should be installed on your virtual machine, but in case you want to run an analysis at home, make sure the commands below run without error (warnings are most likely to be okay). Just as a reminder in case a package is not installed run: `install.packages("vegan", dependencies=TRUE)`.

```
# run withour error
library(adegenet)
library(Rcpp)
library(raster)
library(rgdal)
library(PopGenReport)
```

```
library(dartR)
library(Sunder)
library(ecodist)
library(gdm)
library(GGally)
library(lme4)
```

To make life easier we created some helper function which can be sourced into your session. The code can be found in this days workshop folder. We 'source' in the helper functions via:

```
source("./WEEG/Prac1_Mon/code/helper functions landscape genetics.R")
```

You should now see some functions being available in your Global Environment (click on the Environment tab to check it).

The data sets for this sesssion are located here as well in the subfolder 'data'.

You can check this via:

```
dir("./WEEG/Prac1_Mon/data/")
```

```
## [1] "CD.rdata"        "data.zip"
## [3] "elevation.asc"   "eucs.tif"
## [5] "koalas_locs.csv" "koalas_snps.csv"
## [7] "roads.tif"       "snptable.csv"
```

1. Load your data

In this tutorial we need several kinds of data to run a landscape genetics analysis.

- a genetic data set (SNPs by individuals)
- coordinates of the individuals
- maps of interest

To make the whole tutorial more exiting we created a data set that resembles a real world scenario (admittedly the dataset is simulated to make sure if behaves fairly nicely).

Koala dataset

Is there an effect of roads, eucalypt density and elevation on the population structure of koalas in the Katoomba area?
The data set consists of a sample of 20 animals of koalas sampled around Katoomba.
The samples have been genotyped and produced 30000 genetic markers (SNPs). For each sample the coordinates were recorded (using Map Grid Australia 94 (=UTM) as the coordinate system).
Using your GIS skills you were able to source three different maps of the Katoomba area. Each map is a raster data set and covers 1000 x 1000 pixel, whereas each pixel has a dimension of 16x16m. Luckily the coordinate system between your samples and the maps match [otherwise you would need to know how to reproject your data sets. At the end of this tutorial you find some short scripts that explain how to do that].
The first map is an digital elevation map of the area in meters [from 110-1160 m]. The second map represents the road network and has different values for highway, normal roads and path/tracks. The third and final map shows the eucalypt density of of koala food trees of the area for each pixel [from 0 to 1].
Your task in this tutorial is to find out which 'landscape' has an effect on the population structure of the koala population.

Genetic data

We will start with the genetic data set. The aim here is to create a so-called genlight object [described in detail in the '`adegenet` package and in the dartR Handbook in the literature folder of todays' workshop] when loading your genetic data. Why genlight? - because it is very compact and allows to handle fairly large SNP data set [up to 1 million SNPs it works well, though some function become slow, e.g. do not try to plot a million snps].

There are several ways to import your data set. The most basic is starting from a text (csv) file, which has the following format: individuals (samples) in rows [including a label in the first column], and loci(snps) in columns [including a label in the first row=header]. See the example of a small snp data set below:

```
snps.data <- read.csv("./WEEG/Prac1_Mon/data/snptable.csv")
kable(snps.data)
```

ind	X1	X2	X3	X4	X5	X6	X7	X8	X9
ind_1	1	2	0	1	2	2	1	1	0
ind_2	1	0	2	1	2	2	2	2	2
ind_3	2	1	2	1	2	0	1	0	1
ind_4	0	2	2	2	2	1	1	2	2
ind_5	1	2	1	1	0	0	1	0	2
ind_6	1	1	1	2	1	2	0	2	0
ind_7	1	0	1	0	1	1	1	0	1

The coding is like 0=homozygote reference, 1=heterozygote and 2=homozygote of the alternate or in other words the entry is the frequency of the alternate allele.

To convert our snps into a genlight object we simply use the `new` function from package `adegenet`. We just need to make sure we strip the first column for the genetic data set and use that column for the names of the individuals (samples), the header (execpt the first column) for the loci names, and finally let the function know that each sample is from an diploid individual.

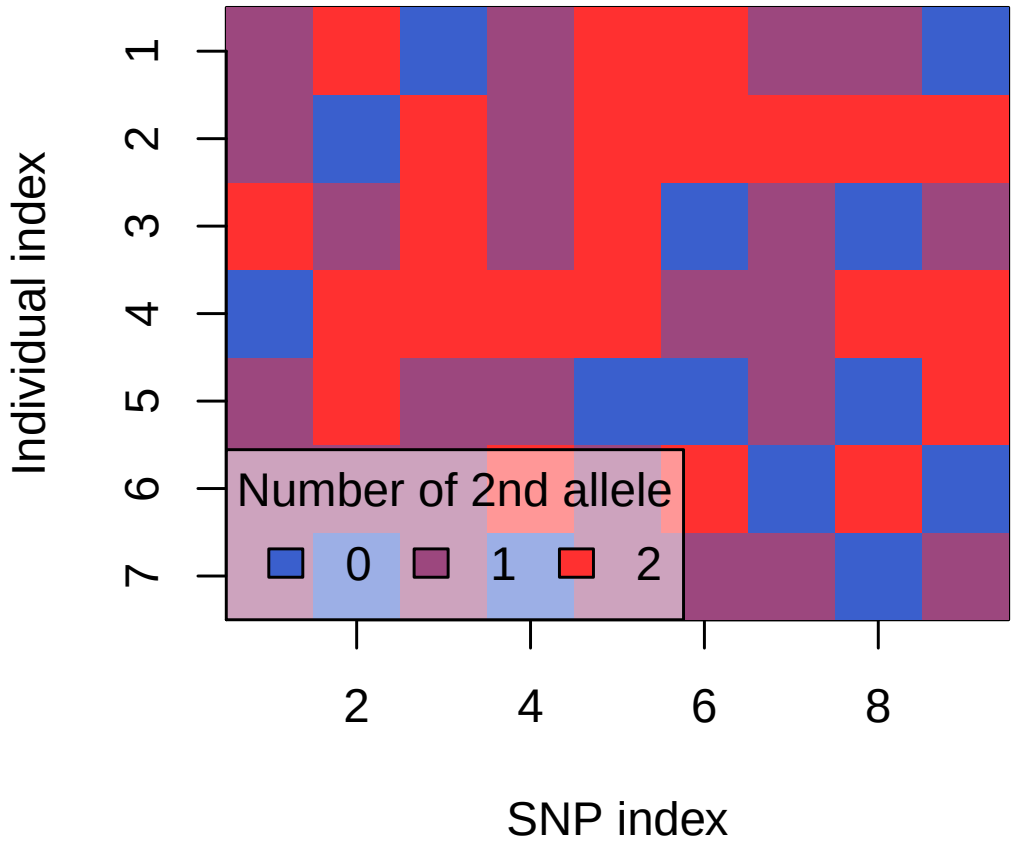
```
snps.gl <- new("genlight", gen = snps.data[, -1],
    ind.names = snps.data[, 1], loc.names = colnames(snps.data)[-1],
    ploidy = rep(2, nrow(snps.data)))
```

This creates a so-called genlight object, which has the great advantage that all the data is stored in one place [genetics, metadata(e.g. coordinates)]. Moreover you can now use all the functions provided by packages such as dartR & adegenet to manipulate your data.

```
snps.gl  #gives an overview of the genlight object
```

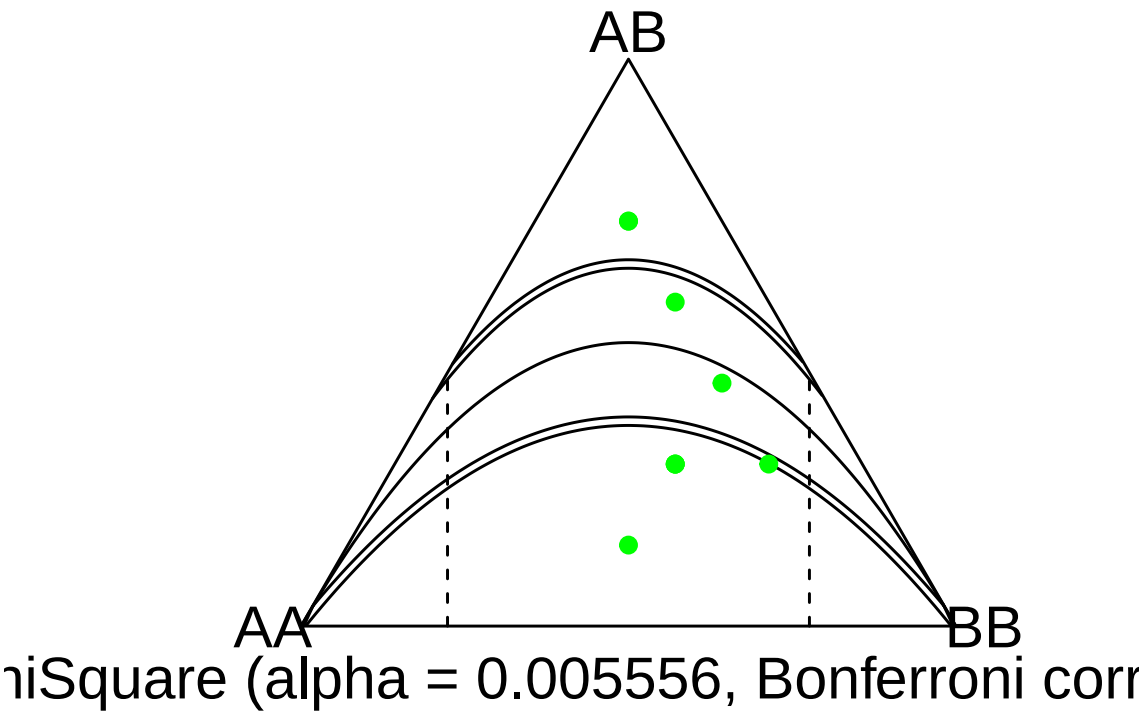
```
##  /// GENLIGHT OBJECT /////////
##
##  // 7 genotypes,  9 binary SNPs, size: 13.5 Kb
##  0 (0 %) missing data
##
##  // Basic content
##    @gen: list of 7 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  7 individual labels
##    @loc.names:  9 locus labels
##    @other: a list containing: elements without names
```

```
plot(snps.gl)
```




```
gl.report.hwe(snps.gl)
```

```
## Starting gl.report.hwe
##   Population assignments not detected, individuals assigned to a single population labelled 'pop1'
##   Calculating HWE for population pop1
```




```
## Reporting significant departures from Hardy-Weinberg Equilibrium
## No significant departures
## Completed: gl.report.hwe
```

```
## [1] Locus  Hom_1  Het    Hom_2  N      Prob
## [7] Sig    BonSig
## <0 rows> (or 0-length row.names)
```

Be aware there are many different ways to create a genlight object from SNP data. If you have received your data from DArT (a service provider for SNP data) you can directly use the gl.read.dart function from package dartR. For vcf data you can use the gl.read.vcf function. For more details check the help pages (e.g. ?gl.read.dart), the vignettes of the packages or the table below [from Gruber et al. 2018: https://onlinelibrary.wiley.com/ doi/full/ 10.1111/1755-0998.12745]:

Import path	Package	Pathway*	Description
gl.read.dart	dartR	—	based on DaRT data [with optional meta data for individuals]
gl.read.csv	dartR	—	csv file of SNP (0,1,2) or A/T, G/C [optional meta data for loci and individual]
read.loci	pegas	loci2genind, gi2gl	data set are provided as a csv text file (?read.loci)
gl.read.vcf	dartR	—	vcf file
read.fstat	adegenet	gi2gl	Fstat format (version 2.9.3) by Jerome Goudet
read.genetix	adegenet	gi2gl	Format Belkhir K., Borsa P., Chikhi L., Raufaste N. & Bonhomme F. (1996-2004) GENETIX
read.structure	adegenet	gi2gl	Structure format of Pritchard, J.; Stephens, M. & Donnelly, P. (2000)
read.PLINK	adegenet	—	Data provided in PLINK format
fasta2genlight	adegenet	—	Extracts SNPs data from fasta format (?adegenet)
read.genetable	PopGenReport	gi2gl	csv text file based on df2genind Adamack & Gruber (2014) (?read.genetable)

*Pathway provides the order of functions needed to convert data to genlight, — indicates that the function direclty converts to a `genlight` object

Task

Task 1

Here comes the first task. In the 'data' folder you find the data set 'koalas_snps.csv'. Load this data set and convert it into a genlight object called `koalas`. The genlight object should have 20 individuals and 30000 loci (SNPs).
In case you get stuck, you can use the hint() function (which is available for every task during the tutorial). Simply type:
`hint(1)`
for this task and you should get some help. In case you want to have the full solution, type:
`solution(1)`
Below you can see the output if you solve it, if you type:
koalas
into the console.

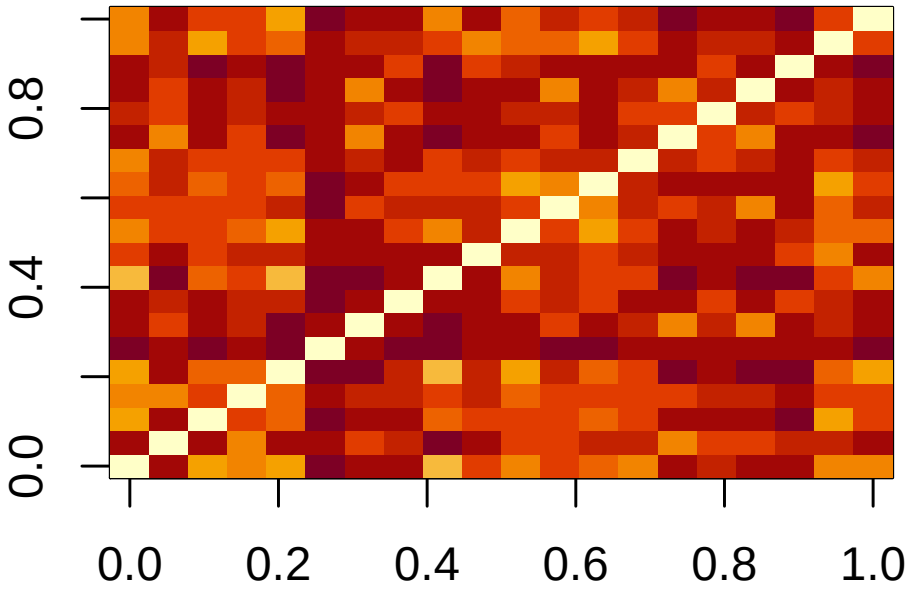
```
##  /// GENLIGHT OBJECT /////////
##
##  // 20 genotypes,  30,000 binary SNPs, size: 2 Mb
##  0 (0 %) missing data
##
##  // Basic content
##    @gen: list of 20 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  20 individual labels
##    @loc.names:  30000 locus labels
##    @other: a list containing: elements without names
```

For our landscape genetic analysis we need to calculate a meassurement of similarity (genetic distance) between the individuals. For our data set we want to use the proportion of shared alleles between each pair of sample [0=no alleles are shared, 1=all alleles are shared between individuals]. For a landscape genetic analysis we actually want to calculate the opposite (a distance) so we use 1-propShared. To do that for all 20 possible pairs (which results in a 20x20 symmetrical matrix) we can use the gl.propShared function, which conviniently uses our genlight object as input:

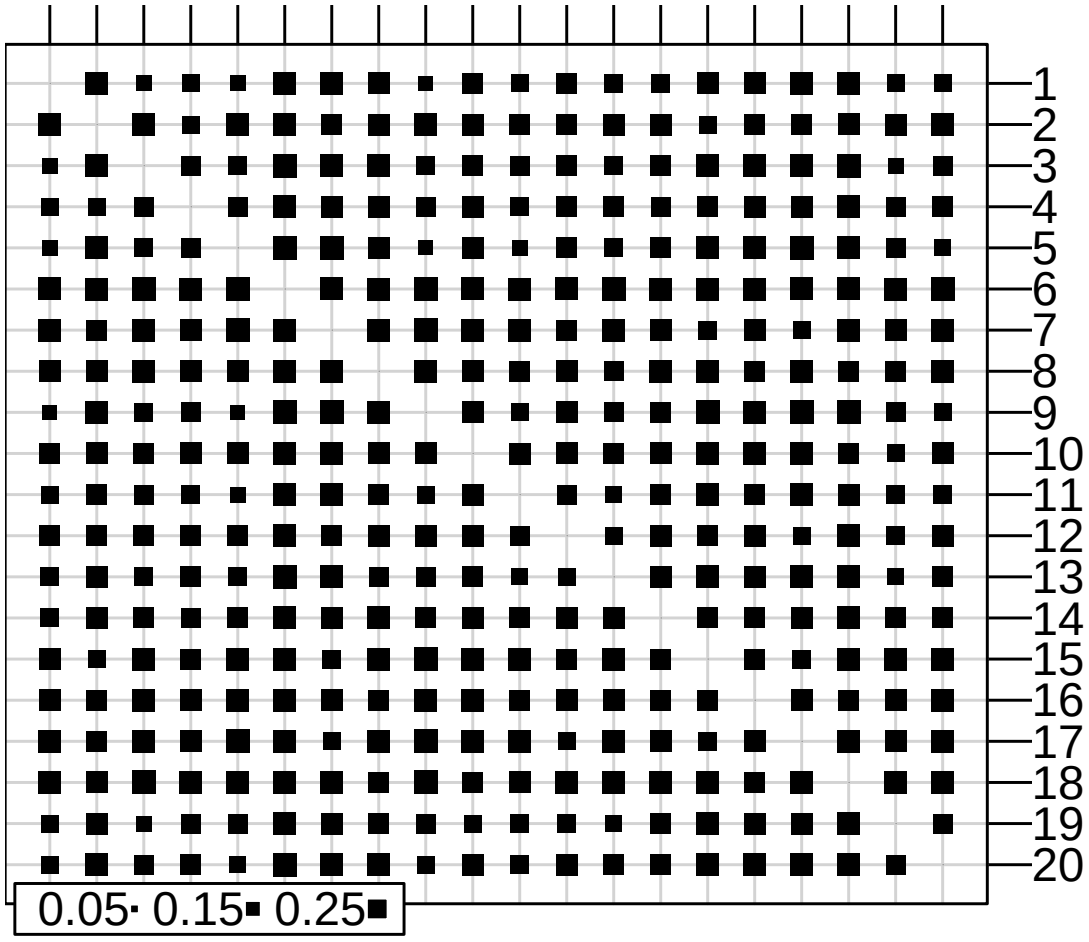
```
# Genetic distance matrix
Gdis <- 1 - gl.propShared(koalas)
```

It is always good to visualise the matrix. There are several commands to do so:

```
image(Gdis)
```




```
table.value(Gdis, csize = 0.4)
```



For example here you can check if you find individuals that are "very" different from the rest, or you can identify clusters of similarity. It would be good if we can check these by comparing pairwise (dis)similarity between individuals and their location, but we have not added coordinates yet…

Hint

Extra Try to understand the structure of the genlight object (koalas). For example nInd(koalas) returns the number of individuals.

Locations

The next step is to attach the locations of your samples to the koala data set. Again there are multiple ways to do it (e.g. gl.read.dart), but we do it in the most basic way, by adding a csv manually to our genlight data set.

The csv file needs to have a column with labels and most importantly you need to know which kind of coordinate system they are in. Most likely (if you recorded your data set via a GPS) you have latitudes and longitudes. [Be aware technically there are different versions of lats and longs, meaning you should know the datum of your coordinate system, which most often is WGS84.]

Lat-longs are a good start but for a landscape genetic analysis you need to convert them into a so-called projected coordinate system, that basically allows to calculate distances in meters [lat-longs are in degrees]. Hence the next three steps are:

- load your coordinates [lat-longs] (and make sure you have a coordinate for each individual)
- attach them to your genlight object
- reproject the lat-longs into a UTM, Zone 56=MGA94 Zone 56 system, which is the most commonly used coordinate system in Australia (best for a limited area, otherwise there are other projections for a continental data set.)
- calculate a pairwise Euclidean distance matrix (=straight line distance, also called geographic distances)

Load your coordinates

The coordinates are stored in the csv file: koala_locs.csv We load them via:

```
latlongs <- read.csv("./WEEG/Prac1_Mon/data/koalas_locs.csv")
head(latlongs)
```

```
##         ind      lon       lat
## 1 koalas_01 150.2850 -33.73710
## 2 koalas_02 150.2448 -33.78967
## 3 koalas_03 150.2439 -33.73189
## 4 koalas_04 150.2850 -33.76870
## 5 koalas_05 150.3153 -33.67394
## 6 koalas_06 150.3943 -33.72850
```

It is always good to check if the labels match and are in the samle order.

```
# check if labels match
sum(indNames(koalas) == latlongs$id)
```

```
## [1] 0
```

Next we store them in our koalas (genlight) object within the slot @other:

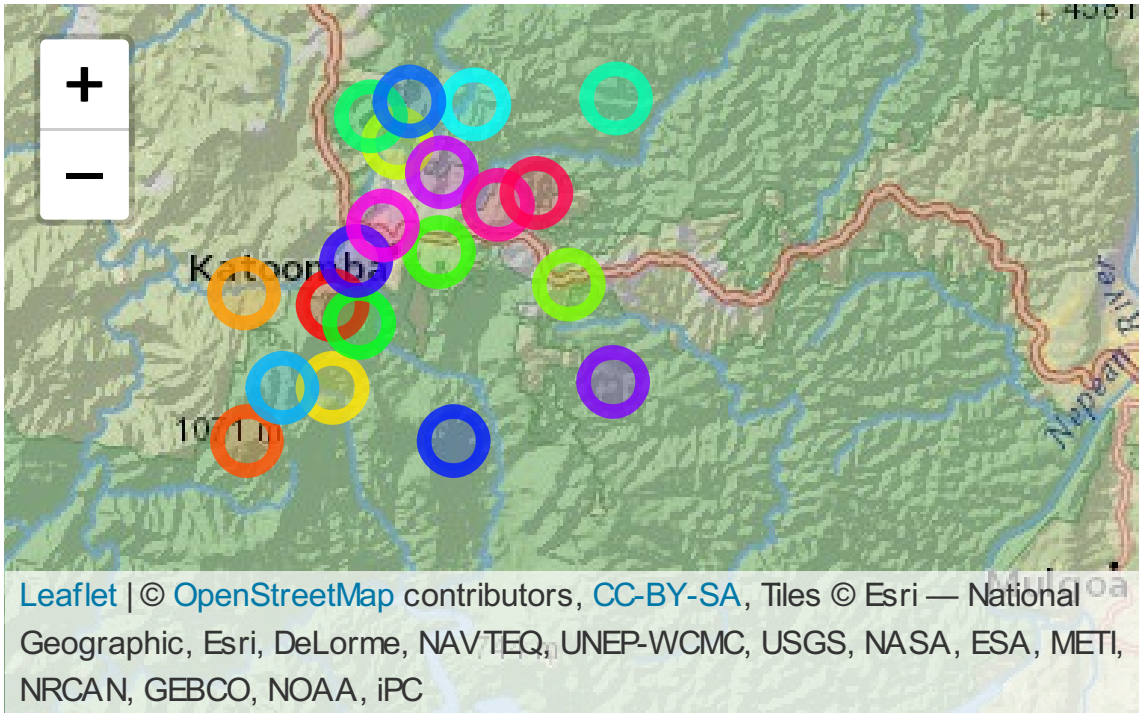
```
koalas@other$latlongs <- latlongs[, 2:3]
head(koalas@other$latlongs)
```

```
##        lon       lat
## 1 150.2850 -33.73710
```

```
## 2 150.2448 -33.78967
## 3 150.2439 -33.73189
## 4 150.2850 -33.76870
## 5 150.3153 -33.67394
## 6 150.3943 -33.72850
```

A nice test if we are at the right spot is the following command:

```
pop(koalas) <- 1:20
gl.map.interactive(koalas)
```



Reproject the coordinates into MGA94, Zone 56.

To be able to project the data set into another format you need to know the definition of the projection. The easiest way to find it, is to search within `spatialreference.org`. Please ask if you are not sure what a "projection" is.

Task

Task 2 Find via google the proj4 string for the projection:
MGA96 Zone 56 [and compare it to UTM 56 South]
at `spatialreference.org`
Then use the `project` function to reproject your latlongs into a new object called `xy`. Be aware that you need to convert the latlongs into a matrix using the `as.matrix` function.
Again you can type `hint(2)` or `solution(2)` if you get stuck.
Compare your coordinates with the output below.

You may noticed that the names of xy are still lat-long (which is not good), so we should change them:

```
colnames(xy) <- c("x", "y")
head(xy)
```

```
##           x       y
## [1,] 248467 6263682
## [2,] 244898 6257752
## [3,] 244641 6264159
## [4,] 248558 6260177
## [5,] 251097 6270761
## [6,] 258576 6264897
```

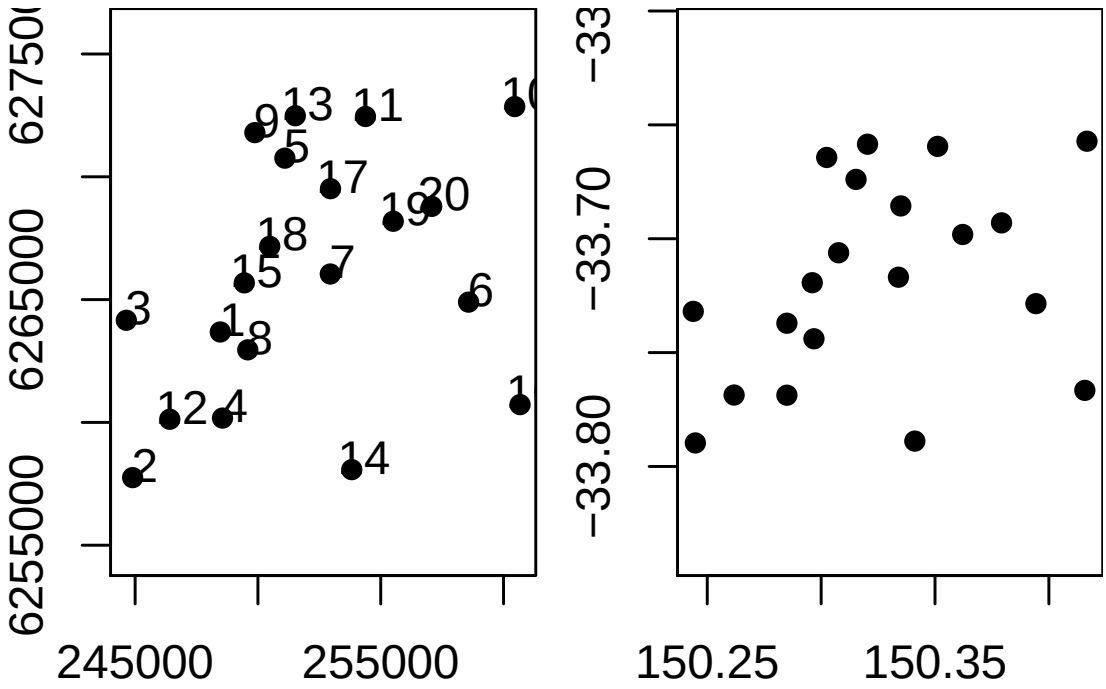
Finally we also store them in our genlight object:

```
koalas@other$xy <- data.frame(xy)
koalas
```

```
##  /// GENLIGHT OBJECT /////////
##
##  // 20 genotypes,  30,000 binary SNPs, size: 2 Mb
##  0 (0 %) missing data
##
##  // Basic content
##    @gen: list of 20 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  20 individual labels
##    @loc.names:  30000 locus labels
##    @pop: population of each individual (group size range: 1-1)
##    @other: a list containing: latlongs  xy
```

Now that we have our coordinates we can calculate the pairwise Euclidean (=geographic, =straigth line) distances between the individuals. Before we do that we plot our data to be able to check our results.

```
par(mfrow = c(1, 2), mai = c(0.5, 0.5, 0, 0))
plot(koalas@other$xy, pch = 16, asp = 1)
text(koalas@other$xy + 500, labels = 1:20)

plot(koalas@other$latlongs, pch = 16, asp = 1)
```



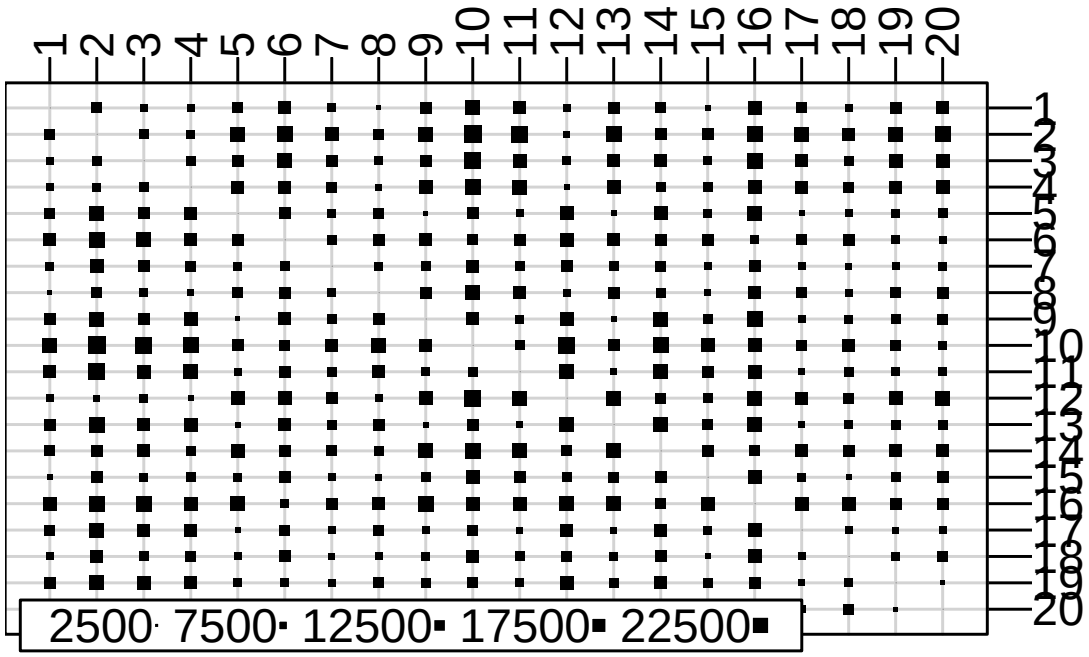
To calculate pairwise distances we can use good old Pythagoras and fortunately we can do it for all pairs in one go.

```
Edis <- as.matrix(dist(koalas@other$xy))
dim(Edis)
```

```
## [1] 20 20
```

The produces a pairwise symmetric distance matrix with dimensions 20x20. As before, when we checked our genetic distance matrix we can visualise it using

```
table.value(Edis, col.labels = 1:20, csize = 0.3)
```



Task

Task 3

a) Find the largest pairwise distance within Edis and check with the plots on the coordinates above.

b) Find the smallest pairwise distance (ignoring the diagonal)

c) Which individual is on average the "most" isolated individual

Now we want to compare our genetic distance matrix (Gdis) with out Euclidean distance matrix (Edis). Basically we do an "Isolation by distance analysis".[1]

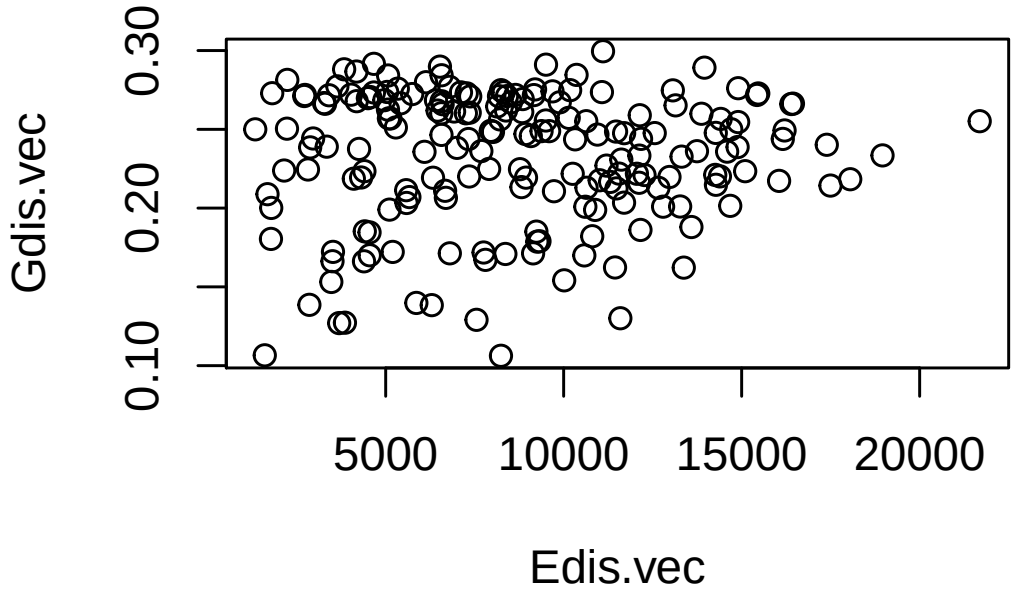
There is one step that is important. First you need to be able to convert your matrix into a vector. There is a convenient function to do so `lower`.

[1] Often this is done on a pairwise population basis plotting log(distance) against Fst/1-Fst. Check ?gl.ibd for more information.

```
Edis.vec <- lower(Edis)
Gdis.vec <- lower(Gdis)
length(Edis.vec)  #Why 190?
```

```
## [1] 190
```

```
plot(Gdis.vec ~ Edis.vec)
```



Do you think there is an isolation by distance effect in our koalas?

Task

Task 4

Run a simple linear regression of Gdis (response) against Edis (predictor). Check the regression coefficient r and the R^2-value.

As you surely remember from the lecture, it is actually not correct to run a simple linear regression on distance matrices (due to non-independence problem of pairwise distances). You need to run a mantel test, which is basically a simple linear regression (resulting in the same r r value), but the p-value (significance) is based on a permutation test, which simple shuffles the distance matrices and check how extreme your r values is compared to the shuffled matrices.

```
ecodist::mantel(Gdis.vec ~ Edis.vec)
```

```
##     mantelr       pval1       pval2
##  0.04668633  0.32200000  0.67900000
```

```
##       pval3   llim.2.5%  ulim.97.5%
##  0.66700000 -0.03651641  0.13550211
```

Be aware we put the package in front of the mantel function as there are several mantel test implementation in R (package vegan has mantel() and mantel.randtest() as functions.)

Compare the output of the mantel test with your linear regression.

Load your maps and quantify your landscape structure

The next and final step before we run our landscape genomic analysis is to load maps, check if they align with the locations (coordinate systems match), convert maps into resistance maps and finally create cost distances from the resistance maps.

We provide three different maps for you to 'play' with.

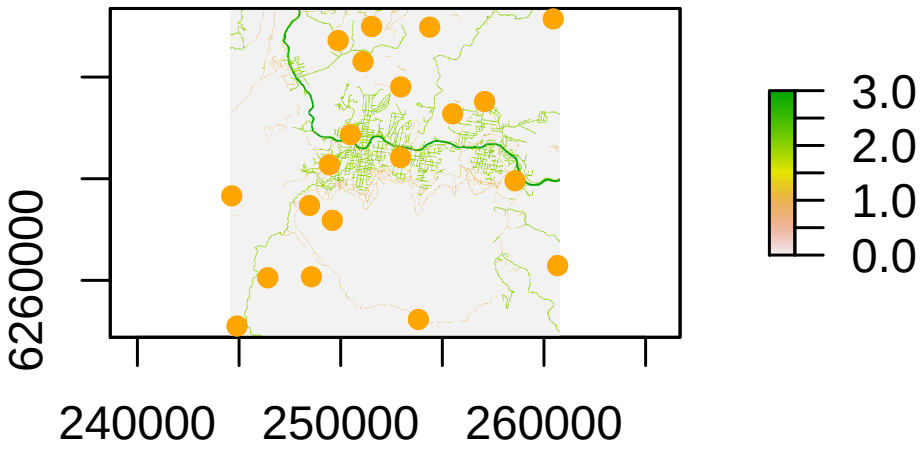
The first map is a map of a road network of the area around Katoomba. We provide it in a geotiff format (which is very convenient as this format comes with a projection[=coordinate system] and it happens to be the same as our xy coordinates, MGA94, Zone 56).

So we simply can load it and check if is "correct", by plotting the sample locations.

```
roads <- raster("./WEEG/Prac1_Mon/data/roads.tif")
roads
```

```
## class       : RasterLayer
## dimensions  : 1000, 1000, 1e+06  (nrow, ncol, ncell)
## resolution  : 16.17666, 16.17666  (x, y)
## extent      : 244591.5, 260768.2, 6257199, 6273376  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=56 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
## data source : D:/Bernd/R/dartRworkshop/WEEG/Prac1_Mon/data/roads.tif
## names       : roads
## values      : 0, 3  (min, max)
```

```
plot(roads)
points(koalas@other$xy, pch = 16, col = "orange")
```



The extent of the map is 1000 by 1000 pixels (each pixel is 16.17 x 16.17 meters [don't ask :-)]) and it is important that the extent of your landscape covers the location of your samples. We can check this via:

```
extent(roads)
```

```
## class       : Extent
## xmin        : 244591.5
## xmax        : 260768.2
## ymin        : 6257199
## ymax        : 6273376
```

```
range(koalas@other$xy$x)
```

```
## [1] 244641 260672
```

```
range(koalas@other$xy$y)
```

```
## [1] 6257752 6272861
```


Task

Extra task
You can try to find a way to "formally" test if all locations are within the extent (there are GIS function in the raster package to do so, e.g. extract, intersect).

If you check closely you can see that the maps "codes" road types differently, depending on their width and the amount of traffic. The main road through Katoomba is coded with a value of 3, the 'normal' roads with a value of 2 and path/tracks have a value of 1. All other pixels are coded as zeros. We can use the `values()` to check which kind of values are used in the raster data set.

```
table(values(roads))  #returns a count on all values in
```

```
##
##      0      1      2      3
## 947000  10908  35716   6376
```

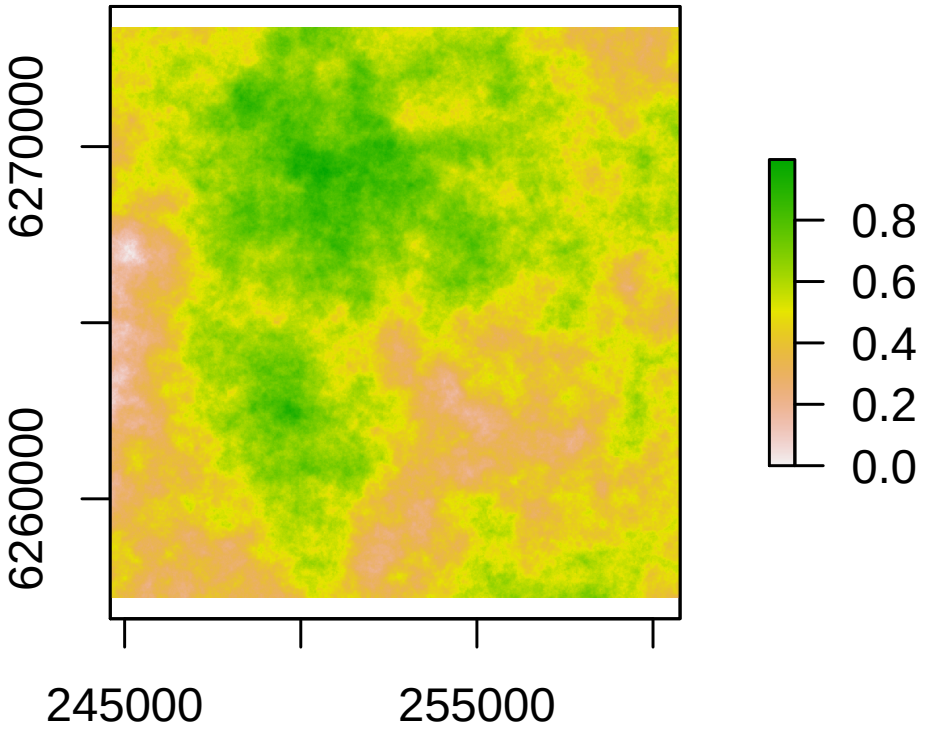
Finally you should also check that the coordinate systems match.

```
crs(roads)
```

```
## CRS arguments:
##  +proj=utm +zone=56 +south +ellps=GRS80
## +towgs84=0,0,0,0,0,0,0 +units=m
## +no_defs
```


Task

Task 5
Load the second may "eucs.tif" and store it under the name "eucs". Plot 'eucs' and check the extent, projection and the "coding". The map is showing the cover of eucalpytus trees within a pixel, so a value of 1 represents a cell that has 100% eucalypt cover and a value of 0 means there are no euclyptus tree in that cell. The idea is that koalas like eucalypts for dispersal and that areas with a high density of eucalypts might promote disperal, whereas no tree limit the dispersal ability of individuals.



The third map 'elevation.asc' is a bit more tricky as it comes as an ascii file (a common format for DEMs (digital elevation models)). This format comes with coordinates but the definition is missing. [2] The first six lines looks like:

```
NCOLS 1000
NROWS 1000
XLLCORNER 244591.520119074
YLLCORNER 6257199.32439938
CELLSIZE 16.176657353316
NODATA_value -3.4e+38
```

You can see that the coordinate system looks good, also the extent and the upper left corner. The NODATA_value is abit odd. As before we load the file using our raster function. [3]

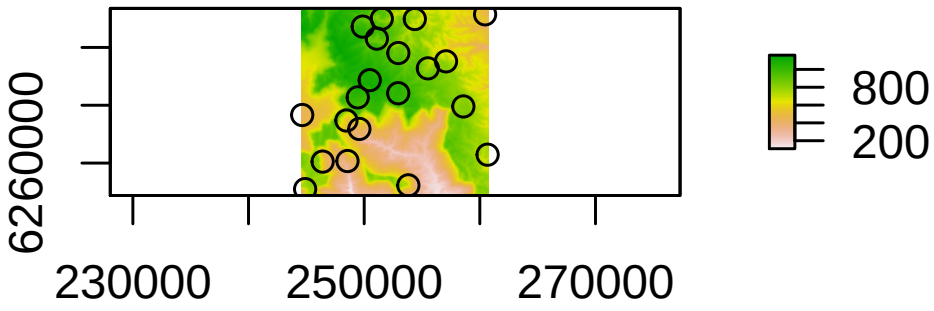
[2] You can check the file, by opening it into a text editor

[3] The raster function is really great it basically allows to load any format into R, alsn png or jpg. So if you want you could draw your landscape in a paint program.

```
ele <- raster("./WEEG/Prac1_Mon/data/elevation.asc")
ele
```

```
## class       : RasterLayer
## dimensions  : 1000, 1000, 1e+06  (nrow, ncol, ncell)
## resolution  : 16.17666, 16.17666  (x, y)
## extent      : 244591.5, 260768.2, 6257199, 6273376  (xmin, xmax, ymin, ymax)
## coord. ref. : NA
## data source : D:/Bernd/R/dartRworkshop/WEEG/Prac1_Mon/data/elevation.asc
## names       : elevation
```

```
plot(ele)
points(koalas@other$xy)
```




```
range(values(ele))  #no missing data!!
```

```
## [1]  110 1160
```

```
crs(ele)
```

```
## CRS arguments: NA
```

All looks great, except the projection. Be aware we need to add the projection to the ele object (otherwise some function will fall over).

```
crs(ele) <- crs(roads)
ele
```

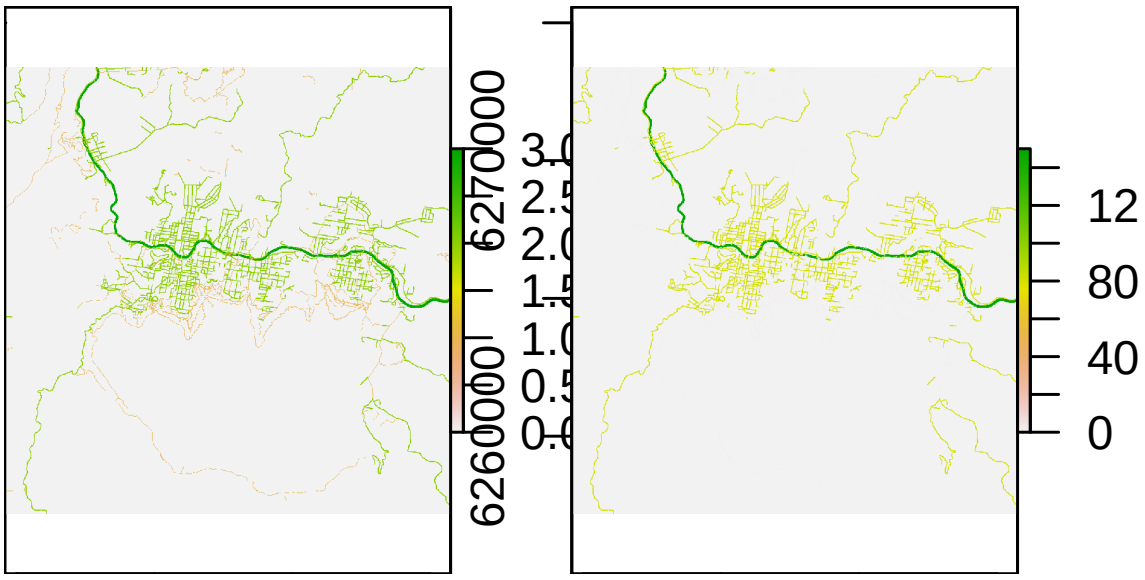
```
## class       : RasterLayer
## dimensions  : 1000, 1000, 1e+06  (nrow, ncol, ncell)
## resolution  : 16.17666, 16.17666  (x, y)
## extent      : 244591.5, 260768.2, 6257199, 6273376  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=56 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
## data source : D:/Bernd/R/dartRworkshop/WEEG/Prac1_Mon/data/elevation.asc
## names       : elevation
```

2. Resistance layers (Quantify your landscape structure)

The next steps are basically all recoding steps. You can imagine that each landscape forms the basis of an hypothesis, namely how it affects the population structure. To operationalise it, we need to translate the maps into a pairwise cost distance matrix. That is basically our idea how the landscape affects the connectivity (dispersal, activity) of individuals. For example we may have the idea that roads limit the ability of koalas to meet each other [especially true for wider roads with more traffic]. So we need to convert pixels of our maps which code for roads with resistance values [the larger the road the higher the resistance value]^. Once done, we then feed that resistance layer to calculate pairwise cost distances between the locations of individuals. There are two major variants (leastcost and circuitscape [=commute in R]). Leastcost is using calculating the shortest path between a pair of locations in terms of resistance values; commute is basically taking all possible paths and averages them in accordance on the resistance values.

Remember our coding is currently:
0: no road
1: path track
2: 'normal road'
3: motorway
So to recode our roads layer we can use the following commands:

```
rec.roads <- roads  #first copy the original layer
rec.roads[values(rec.roads) == 3] <- 150  # a strong effect for motorways
rec.roads[values(rec.roads) == 2] <- 80  #intermediate for roads
rec.roads[values(rec.roads) == 1] <- 1  # a very small effect of tracks
rec.roads[values(rec.roads) == 0] <- 0  # no road has no resistance


# check if recoding worked as expected
par(mai = c(0, 0, 0, 0), mfrow = c(1, 2))
plot(roads)
plot(rec.roads)
```




```
table(values(roads))
```

```
##
##      0      1      2      3
## 947000  10908  35716   6376
```

```
table(values(rec.roads))
```

```
##
##      0      1     80    150
## 947000  10908  35716   6376
```

Once recoded we can calculate the cost distance matrix using the function gl.costdistances(). Please note we will code several cost distance matrices and to keep everything neat and type we collect them into one big list, called CD.

```
CD <- list()  #creates and empty list

# be aware that takes about 2 minutes
system.time(CD$roads <- gl.costdistances(landscape = rec.roads +
    1, locs = koalas@other$xy, method = "commute",
    NN = 8))
```

Task

Task 6 a) Plot Gdis (genetic distances) against CD$roads. b) Run a mantel test between the roads cost distance and genetic distances. Do you think there is an effect of roads?

Why +1? After our recoding we had values of zero for "no road"-areas. But zero values basically means there is no resistance. So an individual would be allowed to connect to any individual across the landscape, if they are connected by pixels that have zero values, regardless how far away they are. Therefore we need to add an offset (+1), so a move of one cell (if there is no resistance due to roads), still costs the "Euclidean distance". [4].

[4] A resistance value of zero, basically would result in an error as there would not be a defined path

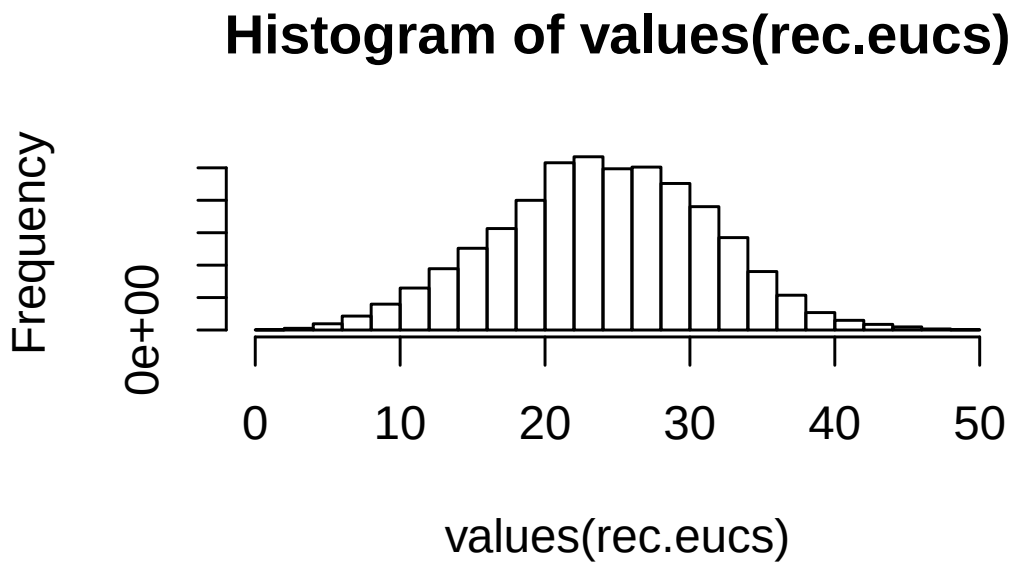
Before we demonstrate the different approaches how to test different cost distances we recode all layers into resistance layer.

`eucs` represents the cover of eucalypts between zero and one. To convert that into a resistance layer we use the formula:

$$(1 - eucs) * 50$$

Why $1 - eucs$? Because this changes a value of 0 to 50 (so a pixel with zero cover has a resistance value of 50). Not as much as a road, but koalas clearly do not like such areas. And a value of 1 turns into zero resistance (again we need to add +1, when we calculate the cost distances)

```
rec.eucs <- (1 - eucs) * 50
hist(values(rec.eucs))
```

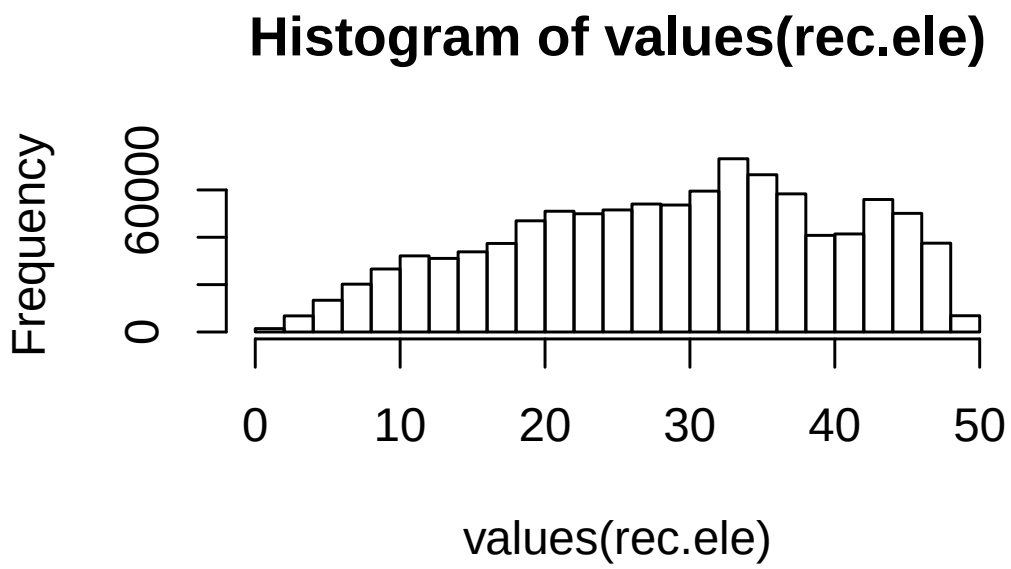


So the cost distance matrix can be calculated via:

```
# another two minutes....
CD$eucs <- gl.costdistances(landscape = rec.eucs +
    1, locs = koalas@other$xy, method = "commute",
    NN = 8)
```

And the next cost distance you want to create is based on elevation (ele). And here it is your choice how to recode it. Basically you might think that elevation has a negative effect (e.g. the higher the elevation, the less likely koalas to disperse). So you could use:

```
rec.ele <- ele

rec.ele <- (rec.ele - min(values(rec.ele)))/diff(range(values(rec.ele))) *
    50
hist(values(rec.ele))
```



The code above standardizes the resistance values of elevation to be between 0-50 (so an elevation of 1160 becomes 50).

To calculate cost distances we use as before:

```
CD$ele <- gl.costdistances(rec.ele + 1, locs = koalas@other$xy,
    method = "commute", NN = 8)
```

Hint

!!! Be aware each cost distance calculation takes about 2-3 minutes. Therefore we prepared a list with all cost distances we came up with and save it under 'CD.rdata' as a file. So if you do not want to wait for the commands below to finish, you can simply type: CD <- readRDS("./WEEG/Prac1_Mon/data/CD.rdata")

The CD object in the hint was created via:

```
CD <- list()
CD$roads <- gl.costdistances(rec.roads + 1, locs = koalas@other$xy,
    method = "commute", NN = 8)

CD$eucs <- gl.costdistances(rec.eucs + 1, locs = koalas@other$xy,
    method = "commute", NN = 8)
CD$ele <- gl.costdistances(rec.ele + 1, locs = koalas@other$xy,
    method = "commute", NN = 8)

CD$eucs.roads <- gl.costdistances(rec.eucs + rec.roads +
    1, locs = koalas@other$xy, method = "commute",
    NN = 8)

CD$eucs.ele <- gl.costdistances(rec.eucs + rec.ele +
    1, locs = koalas@other$xy, method = "commute",
    NN = 8)

CD$ele.roads <- gl.costdistances(rec.ele + rec.roads +
    1, locs = koalas@other$xy, method = "commute",
    NN = 8)
```

Which values you want to attach to a road type or eucalyptus density is basically part of your hypothesis and there are several approaches currently used (test different values= fishing for correlations, resistanceGA(), additional data such as telemetry).

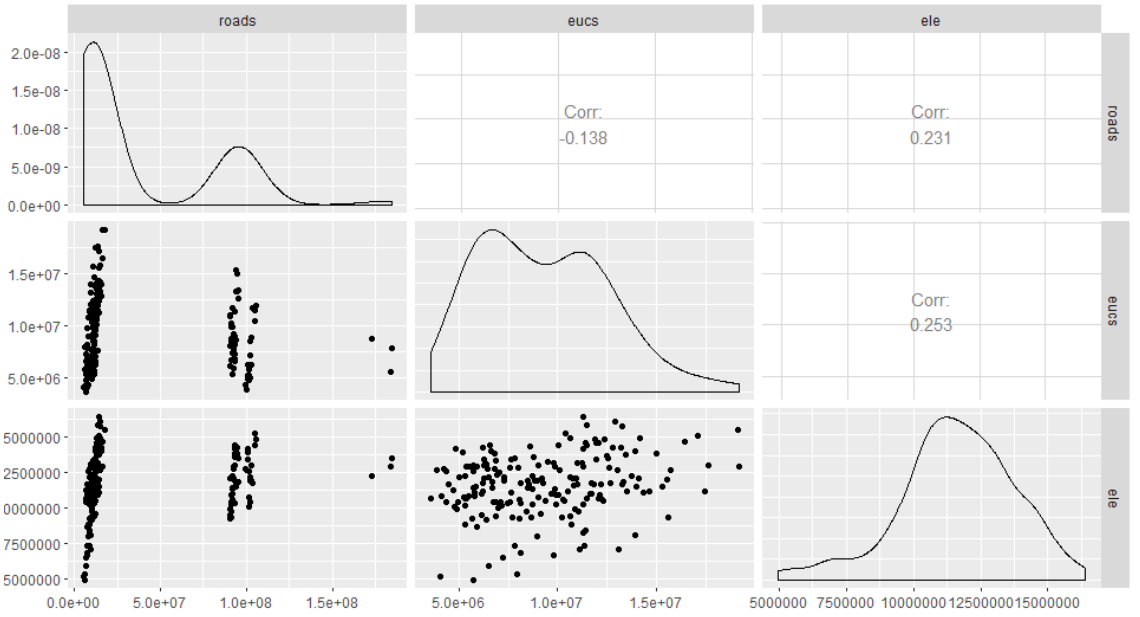
The main caveate is, if you use more than one resistance layer, that the resulting cost distance matrices need to be as different as possible from each other (=low correlation). This is also true for your Null model distance matrix the Eucidean distance matrix, compared to all others.

So let's check if that is true for our three basic cost distances, (roads, eucs and). We use a nice function called ggpairs. Unfortunately it needs some reformating to be usefull. We need to convert our list of cost distance (CD) into a data.frame of vectors. Not a big deal but it looks frightening…

```
names(CD)
```

```
## [1] "roads"      "eucs"       "ele"
## [4] "eucs.roads" "eucs.ele"   "ele.roads"
```

```
ggpairs(data.frame(lapply(CD[c("roads", "eucs",
    "ele")], lower)))
```



So far so good. All of our predictors are not correlated, so we can use all three of them. The usual threshold is if an $|r|>0.7$ then you should not use them in an analysis together.

And here now comes your "creativity" into play. There is nothing that stops us to create as many costdistance matrix as you like (though before you can test them you need to be sure that they are not correlated). To get you used to the idea you can use the following task and create your own hypothesis.

Feel free to use also method "leastcost" in case your hypothesis is that koalas do know their landscape very well and "plan" their movement using the leastcost path approach.

What do you think is "behind" the commute approach?

Read the helppages of ?rSPDistance as this is another version to calculate cost distances (and yet to be explored).

Task

Task 7

Create some resistance matrices

a) Your hypothesis is that eucs and roads both have an effect. Therefore you can simply add the two resistance values together: rec.eucs+rec.roads.

b) Your next hypothesis is that all three resistance layers have an effect. So you add them together, but the maximum cost value should be 100 and minumum cost 0.

c) Create a random resistance (uniform random values between 0 and 50)

d) Create your own (univariate or multivariate) resistance layer from the three available landscape layers.

Once you finished you can calculate the cost distance matrices from the resistance layers using the gl.costdistances() function and store them in the CD object, but read the hint below first, before you spent all afternoon on cost distances.

3. Landscape genetic analysis

Mantel tests

To run a mantel test on all cost distances in turn we can use, the helper function all.mantel from Niko:

```
all.mantels(Gdis, Edis, CD)
```

```
##      Variable   Mantel_r Mantel_p
## [1,] Geo        0.047    0.368
## [2,] roads      0.437    0.003
## [3,] eucs       0.052    0.381
## [4,] ele        0.202    0.077
## [5,] eucs.roads 0.315    0.007
## [6,] eucs.ele   0.139    0.157
## [7,] ele.roads  0.41     0.003
##      partMantel_r partMantel_p
## [1,] <NA>         <NA>
## [2,] 0.45         0.003
## [3,] 0.027        0.385
## [4,] 0.205        0.09
## [5,] 0.343        0.007
## [6,] 0.164        0.117
```

```
## [7,] 0.409        0.003
```

!!NOTE: This function only reports the p-value for the hypothesis that the true relationship between genetic and landscape distances is HIGHER than a random association. Thus, the function assumes that you have coded your resistance layers in a way that leads to positive correlation coefficients! When using this with your own data, make sure to check the sign of the correlations, as you might have high negative correlations that are deemed insignificant in the output of this function." This is also the case for your Wasserman function (see below)!!

Question

Which predictors are good ones?
Which would you keep?
Do we have IBD?
Discuss with your neighbour!!!!
Be aware your answers might be different as you may have created different cost matrices.

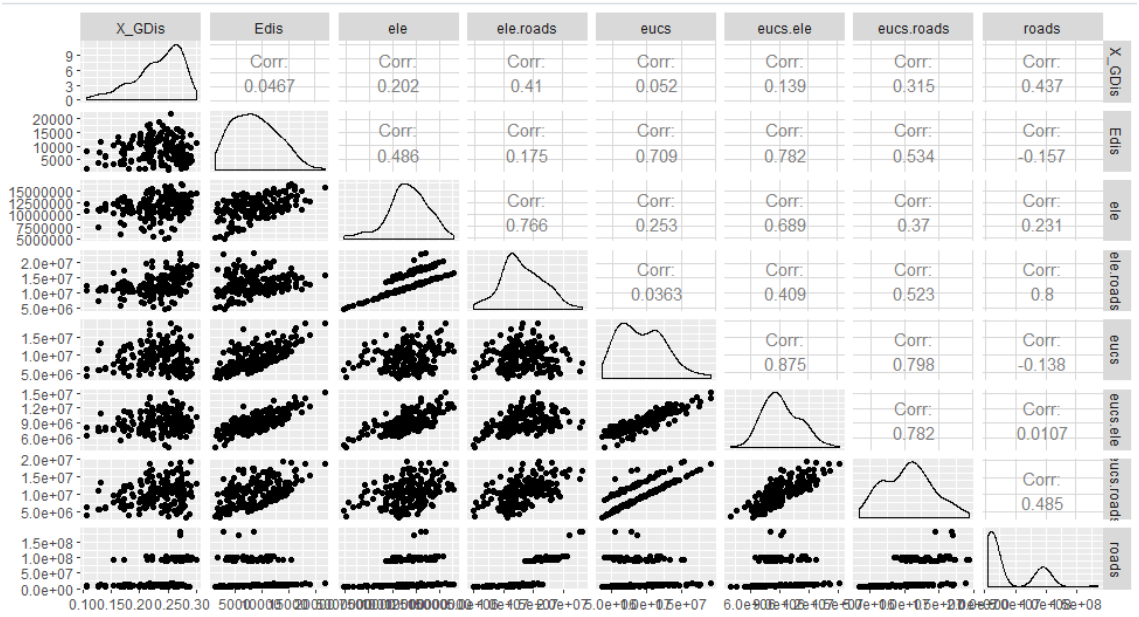
Before we get too excited (e.g. roads [and its combinations] seems to be a good candidate, we need to make sure that they are not too highly correlated. So let use collect all matrices in a single object (we need that later anyway).

```
Alldis <- CD  #copy all costdistances
Alldis$"_GDis" <- Gdis  #add our genetic distance matrix
Alldis$Edis <- Edis  #add Euclidean distance matrix

# sort to have _GDis first [underscore is used
# to make _GDis first]
Alldis <- Alldis[order(names(Alldis))]

names(Alldis)
```

```
## [1] "_GDis"      "Edis"       "ele"
## [4] "ele.roads"  "eucs"       "eucs.ele"
## [7] "eucs.roads" "roads"
```

Now we can use the object Alldis to plot all correlations:

```
ggpairs(data.frame(lapply(Alldis, lower)))
```



As you can see there are quite some correlations higher than 0.7, so we need to remove some of them to get a valid answer in the following analysis.

Partial mantel tests (=causal modelling, invented by Cushman et al. and nicely explained by Wasserman et al.)

Above we have seen that the correlations between the "base"" resistances matrices are quite low, hence we can use those to demonstrate the causal modelling approach. The function we use is wassermann[5]

Test all of them one by one:

[5] Unfurtunately this function uses the wrong spelling from the researcher who nicely explained the approach, but it went undetected and is now published like that. The actualy publication I based my function on was Tzeidle N. Wasserman, Landscape Ecol. 2010, 25, 1601–1612.

```
wassermann(Gdis, CD["eucs"], Edis, plot = FALSE)
```

```
## $mantel.tab
##                    model      r    p
## 1 Gen ~eucs | Euclidean 0.0269 0.41
## 2 Gen ~Euclidean | eucs 0.0139 0.42
```

```
wassermann(Gdis, CD["roads"], Edis, plot = FALSE)
```

```
## $mantel.tab
##                     model      r     p
## 1 Gen ~roads | Euclidean   0.45 0.001
## 2 Gen ~Euclidean | roads 0.1298 0.119
```

```
wassermann(Gdis, CD[c("ele")], Edis, plot = FALSE)
```

```
## $mantel.tab
##                   model       r     p
## 1 Gen ~ele | Euclidean  0.2053 0.074
## 2 Gen ~Euclidean | ele -0.0601 0.668
```

Question

Again which of the base resistance matrices turn out to be important to describe the population structure of koalas?

If you are able to look through it you can also run all the costdistances against each other:

```
wassermann(Gdis, CD[c("eucs", "roads", "ele")],
    Edis, plot = FALSE)
```

```
## $mantel.tab
##                     model       r     p
## 9  Gen ~roads | Euclidean    0.45 0.002
## 2       Gen ~roads | eucs  0.4486 0.001
## 7        Gen ~roads | ele  0.4092 0.005
## 11   Gen ~ele | Euclidean  0.2053 0.064
## 4         Gen ~ele | eucs  0.1954 0.062
## 10 Gen ~Euclidean | roads  0.1298 0.101
## 1       Gen ~eucs | roads  0.1261 0.206
## 8        Gen ~ele | roads  0.1156 0.214
## 5   Gen ~eucs | Euclidean  0.0269  0.44
## 6   Gen ~Euclidean | eucs  0.0139 0.423
## 3         Gen ~eucs | ele   0.001 0.498
## 12   Gen ~Euclidean | ele -0.0601 0.692
```

Maybe a better approach is to use MMRR

```
lgrMMRR(Gdis, CD[c("eucs", "roads", "ele")], Edis)
```

```
## $mmrr.tab
##       layer  coefficient tstatistic tpvalue
## 3     roads 4.339859e-10  6.1479952   0.004
## 2      eucs 9.284315e-10  0.7896475   0.678
## 4       ele 1.323564e-09  0.8454953   0.680
## 1 Intercept 1.905362e-01 11.3390620   0.747
```

```
## 5 Euclidean 2.930316e-07  0.2744262   0.838
##      Fstat Fpvalue        r2
## 3 12.23455   0.009 0.2091929
## 2       NA      NA        NA
## 4       NA      NA        NA
## 1       NA      NA        NA
## 5       NA      NA        NA
```

As you can see here roads come out highly important, but not the other two layers.

But what about using eucs.roads in the mix as well:

```
lgrMMRR(Gdis, CD[c("eucs.roads", "ele")], Edis)
```

```
## $mmrr.tab
##        layer   coefficient tstatistic
## 2 eucs.roads  4.433986e-09   4.649289
## 4  Euclidean -2.417981e-06  -2.809971
## 3        ele  3.482240e-09   2.292073
## 1  Intercept  1.664064e-01  10.293123
##   tpvalue    Fstat Fpvalue        r2
## 2   0.023 10.40922   0.031 0.1437555
## 4   0.081       NA      NA        NA
## 3   0.279       NA      NA        NA
## 1   0.974       NA      NA        NA
```

Feel free to test your layers, again make sure they are not highly correlated.

As an example we know the correlation between eucs and eucs.ele is >0.7 and so between eucs and eucs.roads. So if we simply throw in all the cost distances without checking we get:

```
lgrMMRR(Gdis, CD[], Edis)
```

```
## $mmrr.tab
##        layer   coefficient tstatistic
## 4        ele  1.048928e-07  1.7252418
## 6   eucs.ele -1.862385e-07 -1.6303347
## 5 eucs.roads  6.126836e-08  1.8917536
## 8  Euclidean  1.867748e-06  1.2105656
## 2      roads -1.508927e-09 -1.2567755
## 7  ele.roads -2.187004e-08 -0.9300545
## 1  Intercept  1.916124e-01 11.0524769
## 3       eucs  2.239396e-08  0.4739081
##   tpvalue    Fstat Fpvalue        r2
## 4   0.333 7.762082   0.021 0.2299053
## 6   0.340       NA      NA        NA
## 5   0.363       NA      NA        NA
```

```
## 8   0.453       NA      NA        NA
## 2   0.539       NA      NA        NA
## 7   0.657       NA      NA        NA
## 1   0.763       NA      NA        NA
## 3   0.793       NA      NA        NA
```

So no layer is significantly contributing in explaining the population structure. Hence as mentioned above make sure your hypothesis are different enough to not being correlated.

Commonality analysis (Prunier et al. 2017)

Similar to MMRR is the commonality anlysis. The good news is that we already have all that we need to run this analysis. Alldis is a list of distance matrices. Most importantly the fist in the list needs to be your genetic distance matrix.

```
names(Alldis)
```

```
## [1] "_GDis"      "Edis"       "ele"
## [4] "ele.roads"  "eucs"       "eucs.ele"
## [7] "eucs.roads" "roads"
```

```
## let us run the base resistances (this time
## by numbers) _Gdis, Edis, ele, eucs, roads
ca.out <- CAmrdm(Alldis[c(1, 2, 3, 5, 8)], regrnperm = 999,
    bootn = 100, bootprop = 0.25)
```

```
## Loading required package: fmsb
```

```
## Loading required package: yhat
```

```
## Warning: package 'yhat' was built under R
## version 3.6.1
```

```
## Registered S3 methods overwritten by 'yacca':
##   method            from
##   plot.cca          vegan
##   print.cca         vegan
##   print.summary.cca vegan
##   summary.cca       vegan
```

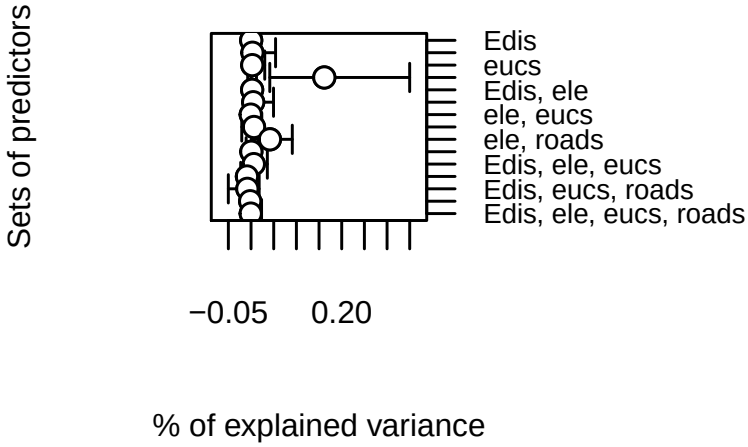
```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

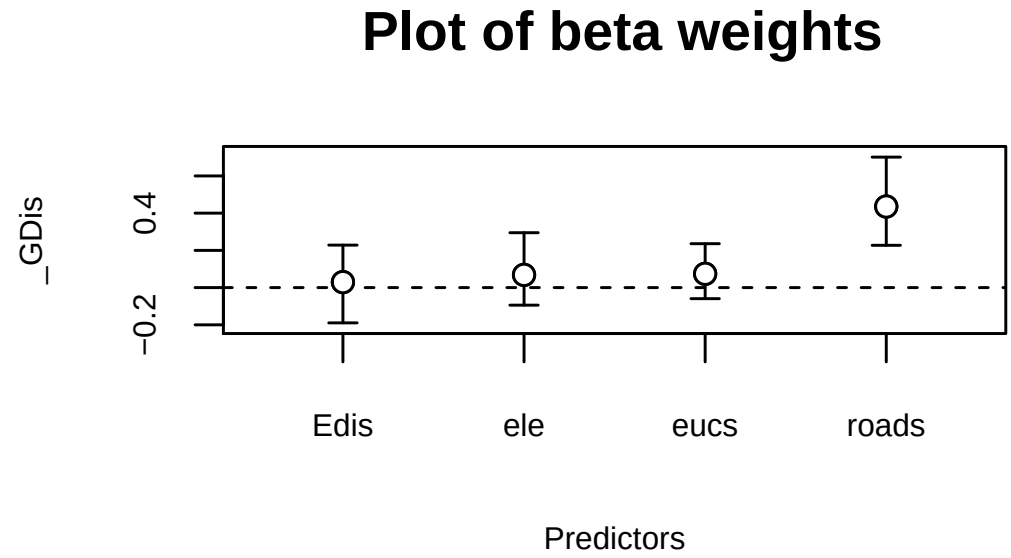
```
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
```

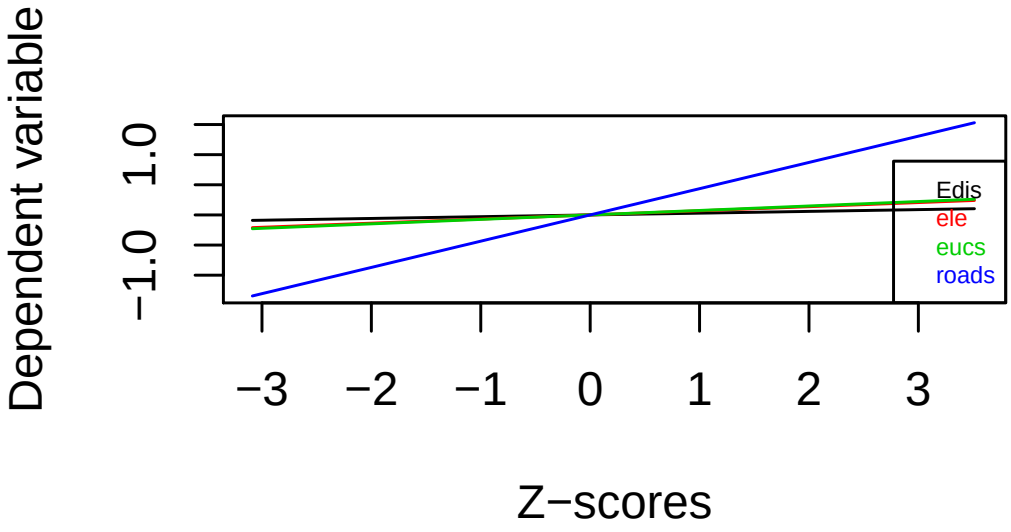
```
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
## [1] 81
## [1] 82
## [1] 83
## [1] 84
## [1] 85
## [1] 86
## [1] 87
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
```

```
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
```

Plot of commonaliti








```
## $Matrixcorrelation
##            _GDis       Edis       ele
## Edis  0.04668633  1.0000000 0.4859909
## ele   0.20194314  0.4859909 1.0000000
## eucs  0.05204557  0.7092583 0.2528497
```

```
## roads 0.43653244 -0.1572076 0.2307813
##             eucs      roads      VIF
## Edis   0.7092583 -0.1572076 2.681982
## ele    0.2528497  0.2307813 1.531338
## eucs   1.0000000 -0.1381202 2.058527
## roads -0.1381202  1.0000000 1.174361
##
## $modelfit
##          R2        pval
## 0.209192889 0.001001001
##
## $suppression
##        %Y %modelfit
## [1,] 2.05  9.799235
##
## $regroutput
##                r    rs      betas
## Edis  0.04668633 0.102 0.02938346
## ele   0.20194314 0.442 0.06840631
## eucs  0.05204557 0.114 0.07407318
## roads 0.43653244 0.954 0.43559585
##             CIinf     CIsup        pval
## Edis  -0.18960983 0.2282027 0.769769770
## ele   -0.09437707 0.2948306 0.426426426
## eucs  -0.05937687 0.2358438 0.453453453
## roads  0.22726480 0.7011443 0.001001001
##       Unique Common  Total
## Edis  0.0003 0.0019 0.0022
## ele   0.0031 0.0377 0.0408
## eucs  0.0027 0.0000 0.0027
## roads 0.1616 0.0290 0.1906
##
## $commonalities
## NULL
```

```
ca.out
```

```
## $Matrixcorrelation
##            _GDis       Edis       ele
## Edis  0.04668633  1.0000000 0.4859909
## ele   0.20194314  0.4859909 1.0000000
## eucs  0.05204557  0.7092583 0.2528497
## roads 0.43653244 -0.1572076 0.2307813
##             eucs      roads      VIF
## Edis   0.7092583 -0.1572076 2.681982
```

```
## ele    0.2528497  0.2307813 1.531338
## eucs   1.0000000 -0.1381202 2.058527
## roads -0.1381202  1.0000000 1.174361
##
## $modelfit
##          R2        pval
## 0.209192889 0.001001001
##
## $suppression
##        %Y %modelfit
## [1,] 2.05  9.799235
##
## $regroutput
##                r    rs      betas
## Edis  0.04668633 0.102 0.02938346
## ele   0.20194314 0.442 0.06840631
## eucs  0.05204557 0.114 0.07407318
## roads 0.43653244 0.954 0.43559585
##             CIinf     CIsup        pval
## Edis  -0.18960983 0.2282027 0.769769770
## ele   -0.09437707 0.2948306 0.426426426
## eucs  -0.05937687 0.2358438 0.453453453
## roads  0.22726480 0.7011443 0.001001001
##       Unique Common  Total
## Edis  0.0003 0.0019 0.0022
## ele   0.0031 0.0377 0.0408
## eucs  0.0027 0.0000 0.0027
## roads 0.1616 0.0290 0.1906
##
## $commonalities
## NULL
```

MLPE (Clark et al. 2002)

To run the MLPE (Maximum likelihood population-effect) mixed models by Clarke et al 2002, we can use different functions provided in the 'resistanceGA' package of Bill Peterman.

Since it is a bit tricky to install this package, we copied and provide the relevant functions from this package via the "helper" file. You should see functions like `MLPElmm` and `ZZ.mat` in your Global Environment tab.

We will use the function `To.From.ID` to create an object that lists all possible pairs of individuals. This object is essentially used to define the covariance structure in the model.

```
ids <- To.From.ID(nInd(koalas))
head(ids)
```

```
##   pop1 pop2
## 1    1    2
## 2    1    3
## 3    1    4
## 4    1    5
## 5    1    6
## 6    1    7
```

The function 'mlpe_rga' requires the lower half of the matrices, so for convenience we can just put all our reformated distances into a dataframe:

```
df <- data.frame(lapply(Alldis, lower))
names(df)
```

```
## [1] "X_GDis"     "Edis"       "ele"
## [4] "ele.roads"  "eucs"       "eucs.ele"
## [7] "eucs.roads" "roads"
```

```
df$pop <- ids$pop1
```

Please note that 'X_GDis' is the name of our genetic distance matrix.

We can now run the different models that we can compare using AIC values, for example the full model with all variables...

```
mlpe.full <- mlpe_rga(formula = X_GDis ~ Edis +
    roads + eucs + ele + (1 | pop), data = df)
```

```
## Warning: Some predictor variables are on very
## different scales: consider rescaling
```

```
summary(mlpe.full)
```

```
## Linear mixed model fit by maximum  likelihood
##  [lmerMod]
##
##      AIC      BIC   logLik deviance df.resid
##   -709.4   -686.7    361.7   -723.4      183
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.0757 -0.3877  0.1690  0.6484  1.7707
##
## Random effects:
##  Groups   Name        Variance  Std.Dev.
##  pop      (Intercept) 0.0001022 0.01011
```

```
##  Residual             0.0011738 0.03426
## Number of obs: 190, groups:  pop, 20
##
## Fixed effects:
##               Estimate Std. Error t value
## (Intercept)  1.880e-01  2.418e-02   7.776
## Edis        -8.141e-08  1.161e-06  -0.070
## roads        4.246e-10  1.024e-10   4.148
## eucs         1.065e-09  1.577e-09   0.675
## ele          1.739e-09  2.231e-09   0.779
##
## Correlation of Fixed Effects:
##       (Intr) Edis   roads  eucs
## Edis   0.569
## roads  0.107  0.212
## eucs  -0.499 -0.604  0.038
## ele   -0.843 -0.607 -0.347  0.127
## fit warnings:
## Some predictor variables are on very different scales: consider rescaling
```

```
AIC(mlpe.full)
```

```
## [1] -709.4498
```

…or a model including only Euclidean distances and effective distances based on roads:

```
mlpe.roads <- mlpe_rga(formula = X_GDis ~ Edis +
    roads + (1 | pop), data = df)
```

```
## Warning: Some predictor variables are on very
## different scales: consider rescaling
```

```
summary(mlpe.roads)
```

```
## Linear mixed model fit by maximum  likelihood
##  [lmerMod]
##
##      AIC      BIC   logLik deviance df.resid
##   -712.5   -696.3    361.3   -722.5      185
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.0665 -0.4265  0.1578  0.6462  1.7440
##
## Random effects:
```

```
##  Groups   Name        Variance  Std.Dev.
##  pop      (Intercept) 0.0001054 0.01026
##  Residual             0.0011776 0.03432
## Number of obs: 190, groups:  pop, 20
##
## Fixed effects:
##              Estimate Std. Error t value
## (Intercept) 2.095e-01  8.869e-03  23.623
## Edis        8.212e-07  6.885e-07   1.193
## roads       4.473e-10  9.654e-11   4.634
##
## Correlation of Fixed Effects:
##       (Intr) Edis
## Edis  -0.705
## roads -0.451  0.082
## fit warnings:
## Some predictor variables are on very different scales: consider rescaling
```

```
mlpe.ele <- mlpe_rga(formula = X_GDis ~ Edis +
    ele + (1 | pop), data = df)
```

```
## Warning: Some predictor variables are on very
## different scales: consider rescaling
```

```
summary(mlpe.ele)
```

```
## Linear mixed model fit by maximum  likelihood
##  [lmerMod]
##
##      AIC      BIC   logLik deviance df.resid
##   -701.3   -685.0    355.6   -711.3      185
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.0521 -0.4365  0.1810  0.6338  1.6006
##
## Random effects:
##  Groups   Name        Variance  Std.Dev.
##  pop      (Intercept) 0.0002563 0.01601
##  Residual             0.0011672 0.03416
## Number of obs: 190, groups:  pop, 20
##
## Fixed effects:
##               Estimate Std. Error t value
```

```
## (Intercept)  1.805e-01  2.732e-02   6.607
## Edis        -8.343e-07  1.056e-06  -0.790
## ele          5.135e-09  2.773e-09   1.852
##
## Correlation of Fixed Effects:
##      (Intr) Edis
## Edis  0.530
## ele  -0.933 -0.733
## fit warnings:
## Some predictor variables are on very different scales: consider rescaling
```

We now can compare the models based on their AIC:

```
AIC(mlpe.roads, mlpe.ele, mlpe.full)
```

```
##            df       AIC
## mlpe.roads  5 -712.5096
## mlpe.ele    5 -701.2735
## mlpe.full   7 -709.4498
```

Sunder (a Bayesian approach)

And the final option is Sunder

```
## Warning takes 'forever' Bayesian approach of
## Botta et al. 2015, which is implemented in
## package 'Sunder':


# Running roads
allel.counts <- gl2sunderarray(koalas)
# setting parameters
nit <- 10^2  ## just for the example, should be much larger, e.g. 50000
run <- c(1, 1, 1)
thinning <- 1  #  just for the example, should be larger, e.g. max(nit/10^3,1)
ud <- c(0, 1, 1, 0, 0)
theta.init <- c(1, 2, 1, 1, 0.01)
n.validation.set <- dim(allel.counts)[1] * dim(allel.counts)[2]/10
theta.max <- c(10, 10 * max(Edis), 10 * max(CD$roads),
    1, 0.01)
plot <- FALSE
trace <- FALSE

# now running the method this is where we use
# the allele counts calculated above
```

```
# (koalas@other$a.counts) the straight line
# distance ('geo') and the effective distance
# created from he resistance surface based on
# roads ('koalas@other$eff.roads') - make sure
# to adjust this input for your own effective
# distances!

sunder.out <- MCMCCV(allel.counts, D_G = Edis,
    D_E = CD$roads, nit, thinning, theta.max,
    theta.init, run, ud, n.validation.set, print.pct = TRUE)

# these calculations will take a bit of
# time...

sunder.out$mod.lik
```

Well done!!

That is the end. Well done you finished. Feel free to have another go or have a well deserved beverage!!!

4. Other bits of code

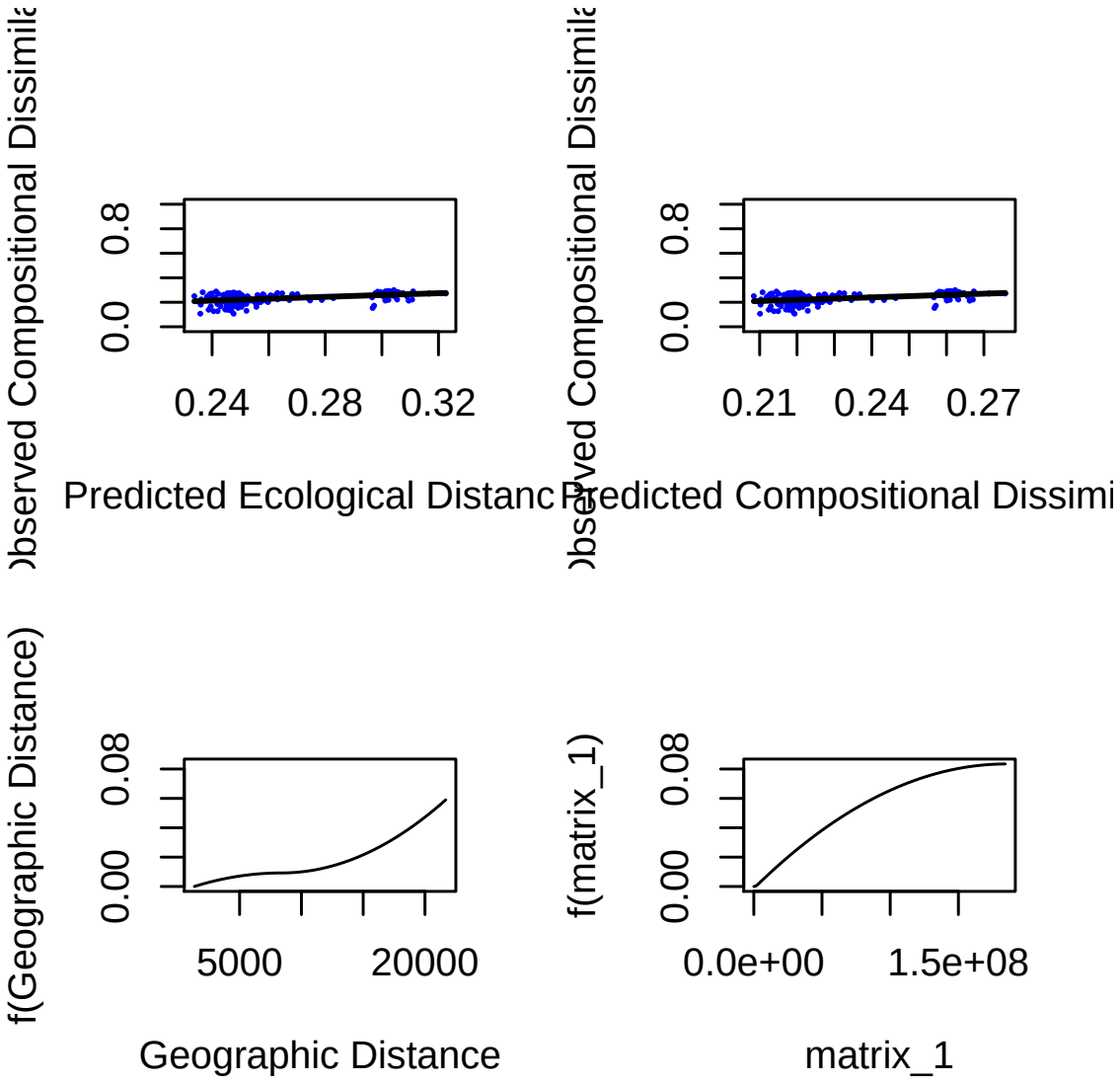
There are some additional functions you might want to have a look at:

Generalised dissimilarity matrices

This is a potentially cool approach, but we have not "played" with it a lot. Please let us know if you have.

Runs a generalised dissimilarity analysis (Kilpatrick & Kellner)

```
gdm <- gl.gdm(xy = koalas@other$xy, gdis = Gdis,
    cdis = CD$roads, geo = T)
```




```
## [1]
## [1]
## [1] GDM Modelling Summary
## [1] Creation Date:  Thu Aug 01 15:45:42 2019
## [1]
## [1] Name:  gdmmodell
## [1]
## [1] Data:  gtab
## [1]
## [1] Samples:  190
## [1]
## [1] Geographical distance used in model fitting?  TRUE
## [1]
```

```
## [1] NULL Deviance:  1.94112019887234
## [1] GDM Deviance:  1.542782188435
## [1] Percent Deviance Explained:  20.521037835202
## [1]
## [1] Intercept:  0.229910606341437
## [1]
## [1] Predictor 1: Geographic
## [1] Splines: 3
## [1] Min Knot: 1335.80836947147
## [1] 50% Knot: 8338.6361845459
## [1] Max Knot: 21687.1677496185
## [1] Coefficient[1]: 0.00923056046328994
## [1] Coefficient[2]: 0
## [1] Coefficient[3]: 0.0505810642918372
## [1]
## [1] Predictor 2: matrix_1
## [1] Splines: 3
## [1] Min Knot: 0
## [1] 50% Knot: 2830270.52764264
## [1] Max Knot: 184366676.288384
## [1] Coefficient[1]: 0
## [1] Coefficient[2]: 0.083438619995616
## [1] Coefficient[3]: 0
## [1]
## NULL
```

Run circuitscape from Windows (to create a current map)

To be able to run this function you need to have Circuitsape installed. !!This is only tested on Windows and will not run on the workshop machines!!

It is assumed Circuitscape is installed at the usual place

`'C:/"Program Files"/Circuitscape/cs_run.exe'`

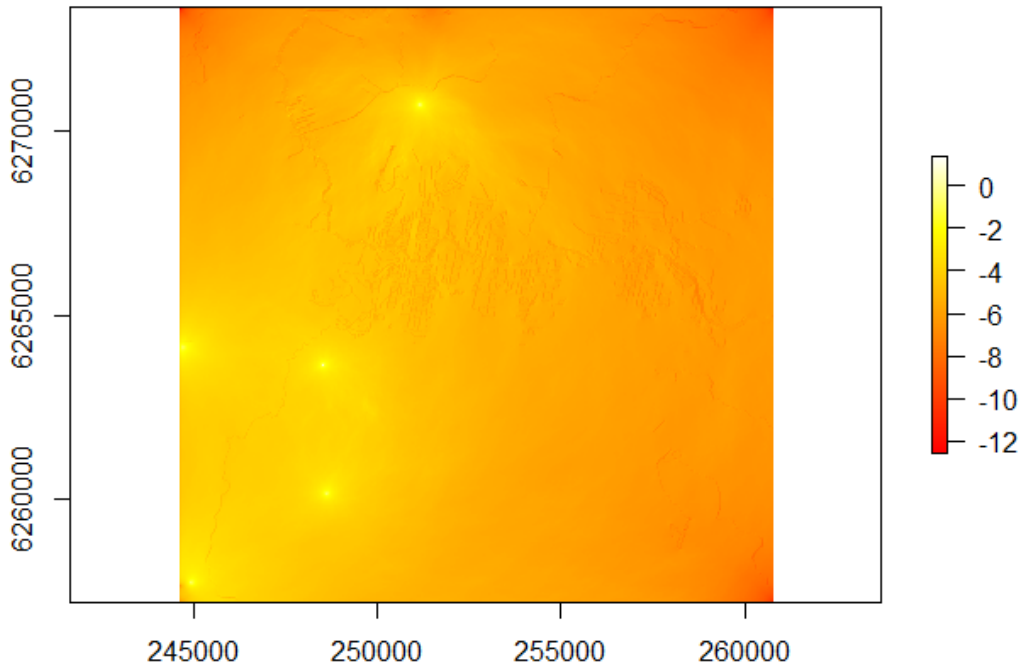
otherwise you need to set the CS_exe parameter. The function returns the circuitscape distance matrix (very similar to commute distances), but maybe more important a circuit current map to visualise the paths.

```
# only the first 5 locations (for testing).
# The whole map takes about an hour to run.
CS <- gl.runCS(landscape = rec.roads + rec.eucs +
    1, outpath = tempdir(), locs = koalas@other$xy[1:5,
    ])
```

For plotting use something like:

```
plot(log(CS$map), col = heat.colors(255))
```

Example output for the first five

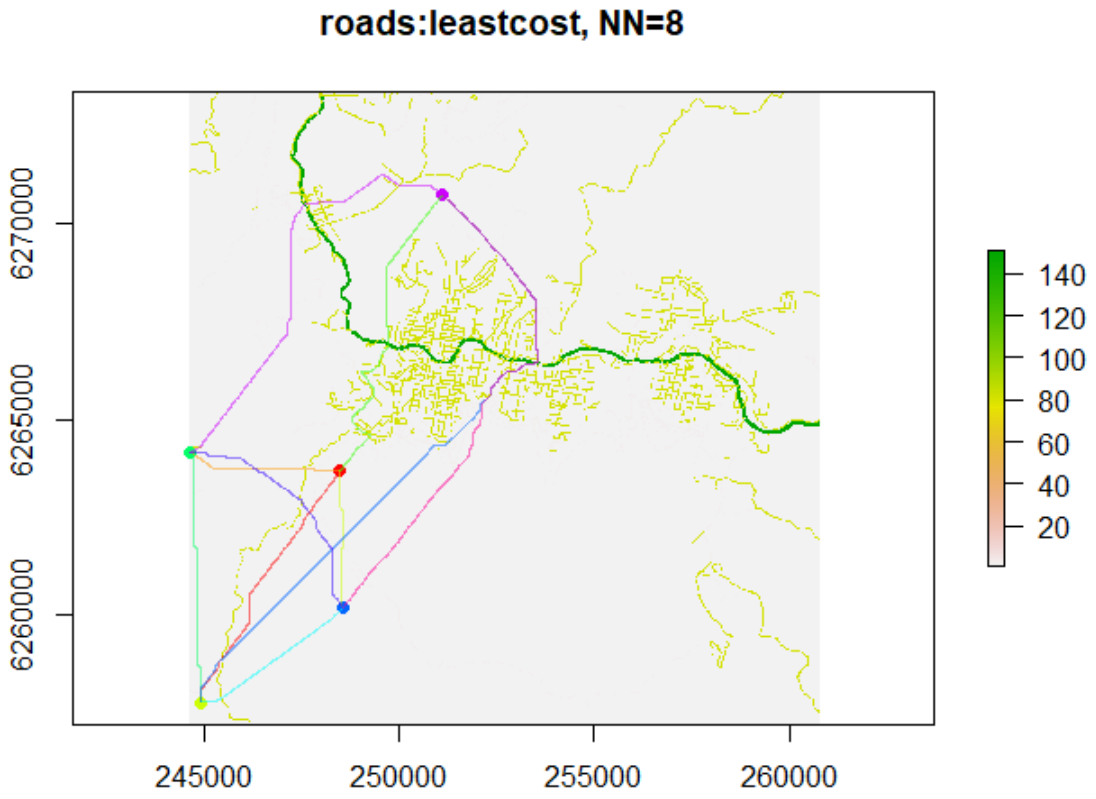


A complete analysis using `gl.genleastcost`

gl.genleastcost is a wrapper function that allows to run a complete analysis in one go (and plots in the case of 'leastcost' the paths on the map).

The output can then be used to run the downstream analysis such as MMRR or commonality analysis. Check ?gl.genleastcost for another example.

```
# takes some minutes to run!!!

glc <- gl.genleastcost(koalas[1:5, ], rec.roads +
    1, "propShared", NN = 8, pathtype = "leastcost")
lgrMMRR(gen.mat = glc$gen.mat, cost.mats = glc$cost.mats,
    eucl.mat = glc$eucl.mat)
```



Reproject your coordinates

From geographical projections (lat-lon) to UTM

You need to know two projections, basically which one your coordinates are in. If you have lat-lon you most likely have coordinates in the WGS84 system. Though be aware that GDA94 (the most common system in Australia) is using a different Datum GRS80, which is slightly different and results in "errors" in the order of 1-2 meters if used unknowingly.

In addition you need to know which your new projection is, for example UTM Zone 55.

We use the project function from rgdal

```
library(rgdal)
canberra <- c(149.13, -35.2809)
hobart <- c(147.3272, -42.8821)

ll <- rbind(canberra, hobart)
ll.sp <- SpatialPoints(ll, CRS("+proj=longlat +ellps=WGS84"))

xy <- spTransform(ll.sp, CRS("+proj=utm +zone=55 +south +ellps=WGS84 +datum=WGS84"))
coordinates(xy)
```

```
##          coords.x1 coords.x2
## canberra  693714.4   6093725
## hobart    526720.5   5252226
```

To backtransform is a bit more difficult as we need to define in which projection we are in.
From UTM to lat-lon

```
utms <- SpatialPoints(xy, CRS("+proj=utm +zone=55 +south +ellps=WGS84 +datum=WGS84"))
ll.sp2 <- spTransform(utms, CRS("+proj=longlat +ellps=WGS84"))
coordinates(ll.sp2)
```

```
##          coords.x1 coords.x2
## canberra  149.1300  -35.2809
## hobart    147.3272  -42.8821
```

```
coordinates(ll.sp)
```

```
##          coords.x1 coords.x2
## canberra  149.1300  -35.2809
## hobart    147.3272  -42.8821
```