

Landscape genetics - Hobart

Bernd Gruber

2019-03-29

Contents

<i>Forword</i>	2
<i>I. Isolation by distance</i>	3
<i>Calculate a genetic distance</i>	3
<i>Calculate a “Euclidean Distance”</i>	3
<i>Isolation by distance</i>	3
<i>II. Landscape genetics using a landscape resistance approach</i>	5
<i>Resistance layer</i>	5
<i>Euclidean distance matrix</i>	6
<i>Cost distance matrix</i>	7
<i>Genetic distance matrix</i>	7
<i>III. A landscape genetics analysis in a single command</i>	8
<i>Load the data</i>	15
<i>IV. A simulated resistance surface landscape analysis</i>	26
<i>A. Simulate a resistance layer</i>	26
<i>B. Populate the landscape with population of a species</i>	27
<i>C. Run simulations and analyze results**</i>	34
<i>V. Create a resistance layer for your own data (a bit experimental!!!)</i>	38
<i>Reproject your coordinates</i>	46

Forword

This tutorial consists mainly of four parts.

- I. The first part demonstrates how to run an isolation by distance analysis.
- II. Then you will learn via an example how you can run a resistance layer approach. Here we explore a relationship between landscape features and their effect expressed as a cost-distance matrix. Very similar to an isolation-by-distance analysis we then use several methods to check for a relationship between cost-distance(s) and genetic distances.
- III. The third part of the tutorial introduces a simulator, which allows you to “play” with the approach and check its power to detect an effect of the landscape. You learn how to simulate landscapes and populate those landscapes with a metapopulation of interest using an individual-based forward simulator.
- IV. The final part allows you to explore the approach using your own data. Either you can use your own data if you have a suitable resistance layer, or you can “create” a resistance layer using a simple graphic program such as Paint to test a “made-up” hypothesis on your data. Obviously you can also skip this part and “play” a bit more with the simulator introduced in the second part.



Hint

You can find an electronic pdf version of this script at the following github page:
<https://github.com/green-striped-gecko/dartRworkshop>

Before we start, a last word on the required data. To be able to work on your own data (part III) you need a genetic data set (SNPs) in the genlight format. There are several ways to achieve this. The table below (see also [<https://onlinelibrary.wiley.com/doi/abs/10.1111/1755-0998.12745>] (Gruber et al. 2018)) you'll see several possible avenues.

In addition you need to have coordinates attached to your samples. Be warned though the tutorial shows you how to project your coordinates into different coordinate systems, you need to learn and understand some basic GIS skills before you can run an analysis with your own data on your own landscape.

We need two main libraries

```
library(dartR)

## Loading required package: adegenet

## Loading required package: ade4

##
##    /// adegenet 2.1.1 is loaded //////////////////////////////////
##
##    > overview: '?adegenet'
##    > tutorials/doc/questions: 'adegenetWeb()'
##    > bug reports/feature requests: adegenetIssues()
```

```
library(PopGenReport)
library(raster)
```

```
## Loading required package: sp
```

I. Isolation by distance

Calculate a genetic distance

Calculate a “Euclidean Distance”

Isolation by distance

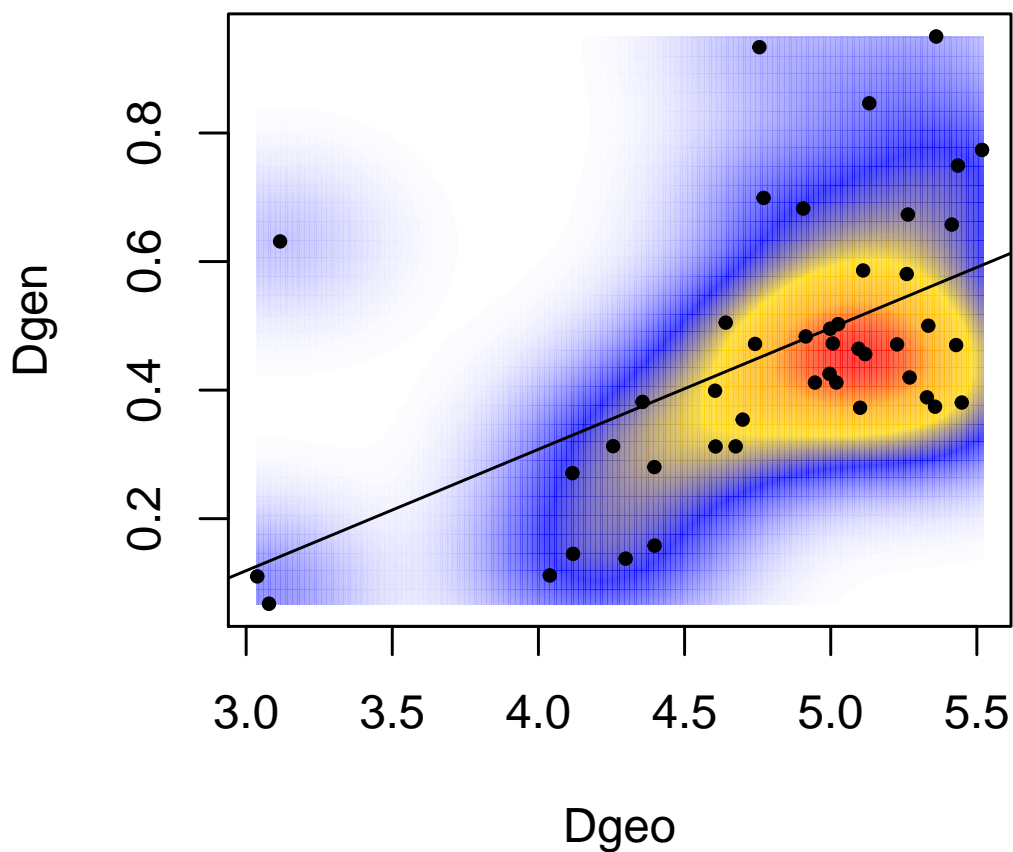
The “null model” in landscape genetics is that there is a simple relationship between genetic and euclidean distance. A standard procedure is to study the relationship between $\log(\text{euclidean distance})$ and $F_{ST}/1-F_{ST}$ (see ?gl.ibd for details). For a quick check we can use the `gl.ibd`. To be able to use the function the genlight object needs to have the coordinates for each individual in the `@other$latlong` slot. Further we need to provide information if the coordinates are already projected or given in lat/lon.

```
iso <- gl.ibd(possums.gl, projected = TRUE)
```

```
## Standard analysis performed on the genlight object. Mantel test and plot will be Fst/1-Fst versus log(dist)
## Coordinates not transformed. Distances calculated on the provided coordinates.
## Mantel statistic based on Pearson's product-moment correlation
##
## Call:
## mantel(xdis = Dgen, ydis = Dgeo, permutations = 999, na.rm = TRUE)
##
## Mantel statistic r: 0.5513
##      Significance: 0.001
```

```
##
## Upper quantiles of permutations (null model):
## 90% 95% 97.5% 99%
## 0.226 0.291 0.340 0.406
## Permutation: free
## Number of permutations: 999
```

Isolation by distance



The function returns a list of the following components: Dgen (the genetic distance matrix), Dgeo (the Euclidean distance matrix), mantel (the statistics of the mantel test)

A mantel test is basically a simple regression, but the significance takes the non-independence of pairwise distances via a bootstrap approach into account.

II. Landscape genetics using a landscape resistance approach

A simulated example

Often ecologists want to know if a particular landscape feature is affecting population structure on top of Euclidean distance. The idea is that a particular feature is causing some cost for individuals when moving through it, hence modifying the actual euclidean distance between individuals/populations. Commonly used approaches to calculate so-called cost-distances are the “least-cost” and “circuitcost” approach. Both approaches require a landscape that represents landscape features in terms of their “resistance” values.

Resistance layer

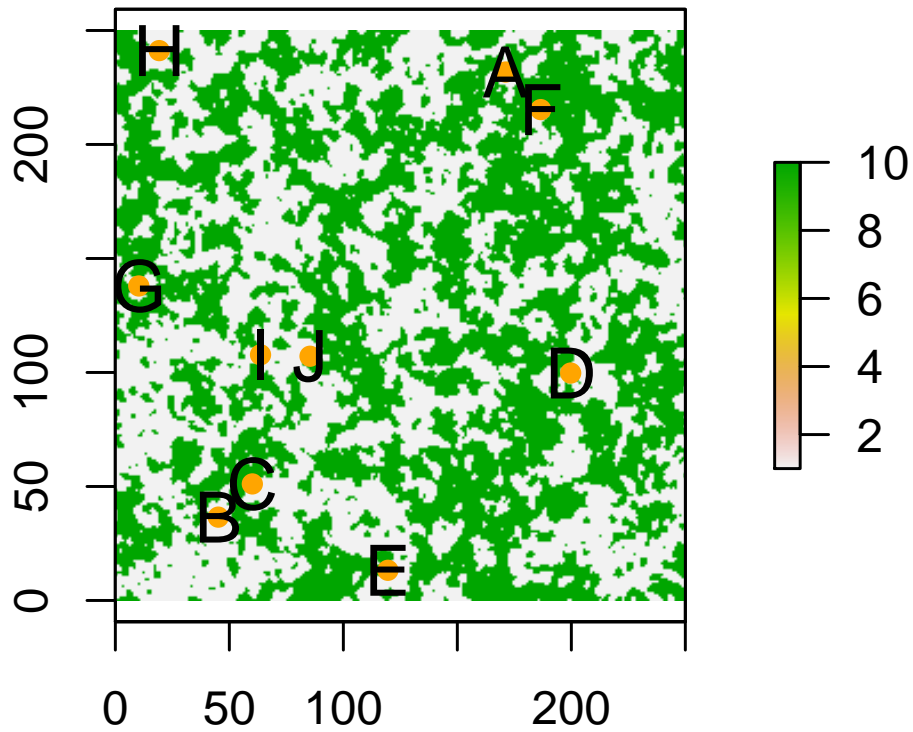
In this example we use populations as the entity of interest. Hence we need to calculate three population-based pairwise distance matrices, namely a Euclidean distance matrix, a cost distance matrix and finally a genetic distance matrix. The two distance matrices can then be used (very similar to the partial mantel test above) to “compete” against each other - how well they explain genetic distances. As mentioned we base our analysis on populations, therefore we first need to calculate the coordinates of our population centers. But first we load our (resistance) landscape to be able to plot our data.

```
landscape.sim <- readRDS(system.file("extdata",
  "landscape.sim.rdata", package = "dartR"))
```

We calculate the population centers via:

```
xs <- tapply(possums.gl$other$latlong[, "lon"],
  pop(possums.gl), mean)
ys <- tapply(possums.gl$other$latlong[, "lat"],
  pop(possums.gl), mean)

plot(landscape.sim)
points(xs, ys, pch = 16, col = "orange")
text(xs, ys, popNames(possums.gl), col = "black",
  cex = 1.5)
```



```
coords <- cbind(xs, ys)
```

Euclidean distance matrix

```
eucl <- as.matrix(dist(coords))
```



Task

1. Explore your Euclidean distance matrix.
2. Note the Euclidean distance between A-F, and I-J.

Cost distance matrix

```
cost <- gl.costdistances(landscape = landscape.sim,
  locs = coords, method = "leastcost", NN = 8)
```



Task

1. Explore your Euclidean distance matrix.
2. Note the cost distance between A-F, and I-J and compare it to the Euclidean distances noted above. Can you see the effect of the landscape between A-F compared to I-J?

Genetic distance matrix

We will use pairwise Fsts between population, but in principle any pairwise genetic distance index can be used.

```
gd <- as.matrix(as.dist(gl.fst.pop(possums.gl,
  nboots = 1)))
```

And finally run a partial mantel test

```
library(PopGenReport)
wassermann(gen.mat = gd, eucl.mat = eucl, cost.mats = list(cost = cost),
  plot = F)
```

```
## $mantel.tab
##           model      r      p
## 1 Gen ~cost | Euclidean 0.4945 0.022
## 2 Gen ~Euclidean | cost -0.2502 0.841
```

```
lgrMMRR(gen.mat = gd, cost.mats = list(cost = cost),
  eucl.mat = eucl)
```

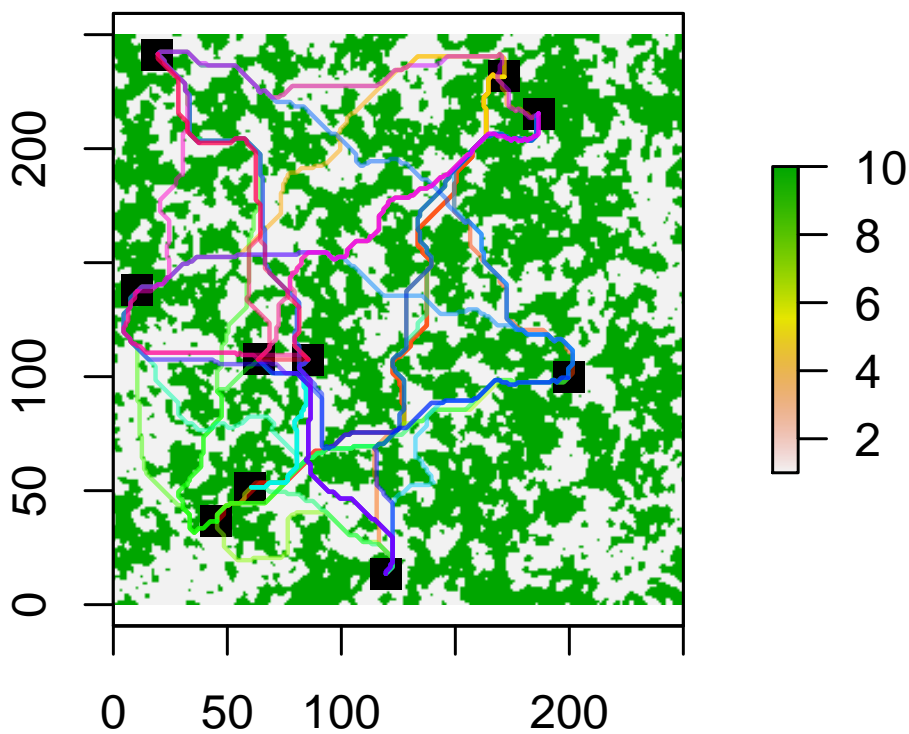
```
## $mmrr.tab
##      layer  coefficient tstatistic tpvalue
## 2      cost 0.0014649146  3.686690  0.062
## 3 Euclidean -0.0009384741 -1.674577  0.322
## 1 Intercept 0.1228057756  4.419399  1.000
##      Fstat Fpvalue      r2
## 2 24.532  0.001 0.5387859
## 3      NA      NA      NA
## 1      NA      NA      NA
```

II. A landscape genetics analysis in a single command

dartR has a convenience function that does it all in once, but is less flexible. It has the benefit that it shows the actual least cost path in the landscape (but runs slower).

```
glc <- gl.genleastcost(possums.gl, fric.raster = landscape.sim,
  gen.distance = "D", NN = 8, pathtype = "leastcost")
```

layer:leastcost, NN=8



##

|
|
|
|
|

| 0%

| 1%

=	2%
=	3%
=	4%
==	4%
==	5%
==	6%
==	7%
===	7%
===	8%
===	9%
===	10%
====	10%
====	11%
====	12%
====	13%
=====	13%
=====	14%
=====	15%
=====	16%
=====	16%
=====	17%
=====	18%

=====	19%
=====	20%
=====	21%
=====	22%
=====	23%
=====	24%
=====	24%
=====	25%
=====	26%
=====	27%
=====	27%
=====	28%
=====	29%
=====	30%
=====	30%
=====	31%
=====	32%
=====	33%
=====	33%
=====	34%
=====	35%
=====	36%

=====	36%
=====	37%
=====	38%
=====	39%
=====	40%
=====	41%
=====	42%
=====	43%
=====	44%
=====	44%
=====	45%
=====	46%
=====	47%
=====	47%
=====	48%
=====	49%
=====	50%
=====	50%
=====	51%
=====	52%
=====	53%
=====	53%

=====	54%
=====	55%
=====	56%
=====	56%
=====	57%
=====	58%
=====	59%
=====	60%
=====	61%
=====	62%
=====	63%
=====	64%
=====	64%
=====	65%
=====	66%
=====	67%
=====	67%
=====	68%
=====	69%
=====	70%
=====	70%
=====	71%

=====	72%
=====	73%
=====	73%
=====	74%
=====	75%
=====	76%
=====	76%
=====	77%
=====	78%
=====	79%
=====	80%
=====	81%
=====	82%
=====	83%
=====	84%
=====	84%
=====	85%
=====	86%
=====	87%
=====	87%
=====	88%
=====	89%

```

===== | 90%
|
===== | 90%
|
===== | 91%
|
===== | 92%
|
===== | 93%
|
===== | 93%
|
===== | 94%
|
===== | 95%
|
===== | 96%
|
===== | 96%
|
===== | 97%
|
===== | 98%
|
===== | 99%
|
===== | 100%

```

```

wassermann(gen.mat = gd, eucl.mat = eucl, cost.mats = list(cost = cost),
  plot = F)

```

```

## $mantel.tab
##           model      r      p
## 1 Gen ~cost | Euclidean 0.4945 0.023
## 2 Gen ~Euclidean | cost -0.2502 0.823

```



Task

Possums in New Zealand

Load the provided data set and the provided raster data set.

Run a resistance layer analysis to check if the proportion of forest in the area do have an effect on the population structure of possums.

We will do this together to demonstrate the “problems” you can run into, when running an analysis.

Load the data

We will use a “real” data set from Possums in NZ (and the Australian mainland). Because of time limitation we will only run a subset of the data.

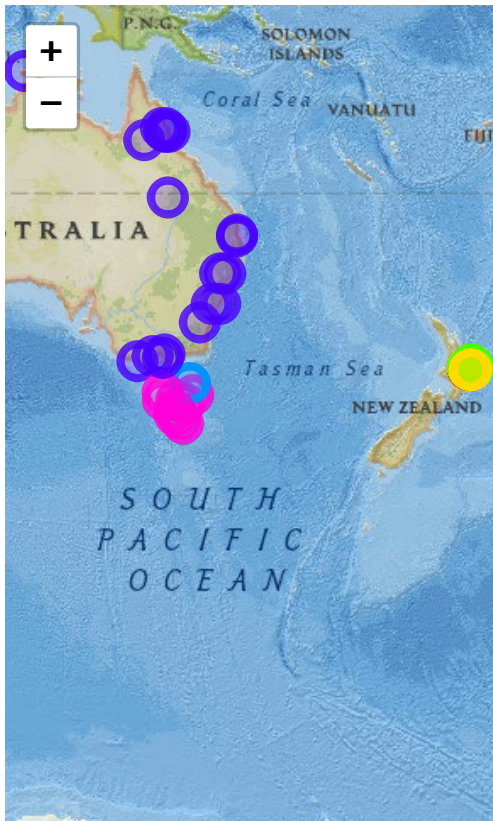
```
# source('https://raw.githubusercontent.com/green-striped-gecko/dartRworkshop/master/code/lgfun.r')
```

```
poss <- gl.read.dart("./data/Possums_SNP.CSV",
  "./data/Possum_Covariate_DArT_180619.csv",
  probar = FALSE)
```

```
## Topskip not provided. Guessing topskip...
## Set topskip to 6 . Proceeding ...
## Trying to determine if one row or two row format...
## Found 2 row(s) format. Proceed...
## Added the following covmetrics:
## AlleleID AlleleSequence SNP SnpPosition CallRate OneRatioRef OneRatioSnp FreqHomRef FreqHomSnp FreqHets P
## Number of rows per Clone. Should be only 2 s: 2
## Recognised: 283 individuals and 9676 SNPs in a 2 row format using ./data/Possums_SNP.CSV
## Start conversion....
## Format is 2 rows.
## Please note conversion of bigger data sets will take some time!
## Once finished, we recommend to save the object using save(object, file="object.rdata")
## Try to add individual metadata: ./data/Possum_Covariate_DArT_180619.csv .
## Ids for individual metadata (at least a subset of) are matching!
## Found 281 matching ids out of 283 ids provided in the ind.metadata file. Subsetting snps now!.
## Added pop factor.
## Added latlon data.
## Added id to the other$ind.metrics slot.
## Added pop to the other$ind.metrics slot.
## Added lat to the other$ind.metrics slot.
```

```
## Added lon to the other$ind.metrics slot.
## Added sex to the other$ind.metrics slot.
## Added coat to the other$ind.metrics slot.
## Read depth calculated and added to @loc.metrics slot.
## All read in. Please check carefully the output above
```

```
gl.map.interactive(poss)
```



Leaflet | © OpenStreetMap contributors, CC-BY-SA, Tiles © Esri — National Geographic, Esri, DeLorme, NAVTEQ, UNEP-WCMC, USGS, NASA, ESA, METI, NRCAN, GEBCO, NOAA, IPC

```
# subset to only NZ data using longitude
poss.nz <- poss[poss@other$latlong$lon > 160,
]
```

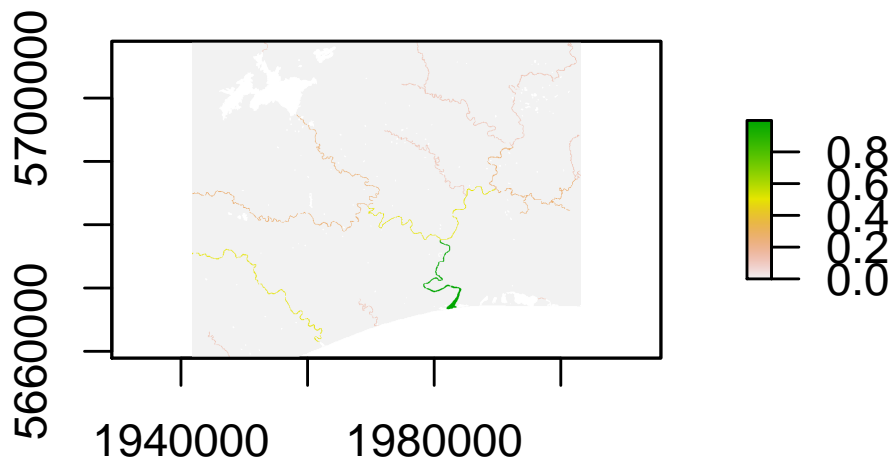
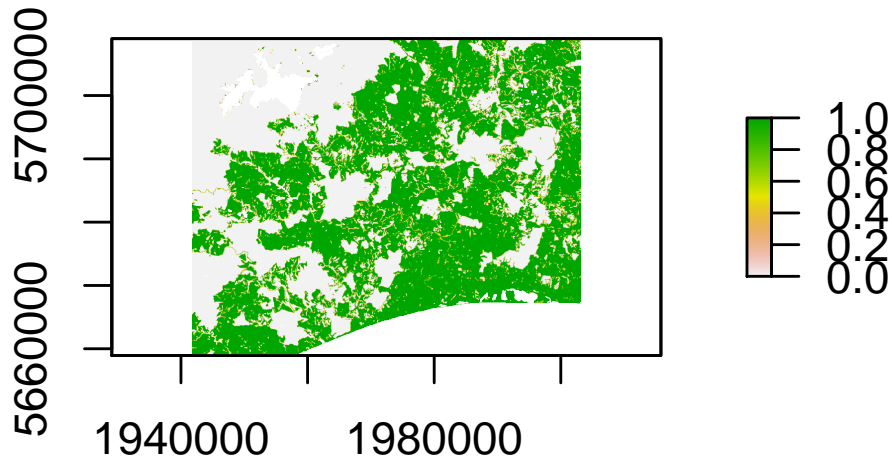
If you inspect the `genlight` object, you notice coordinates are not projected (only lat-lons are

given). We need to project the data set into the same format as the resistance layers. So we first load the resistance layers and check their coordinate system.

The resistance layers (ts and rb) code for the amount of trees and shrubs in a grid cell and for rivers and bridges in the landscape. We want to test if the amount of forest/trees or rivers/bridges affect the population structure of possums in our study area. From other studies it is known that possums prefer to disperse within forest, so we expect a positive relationship between amount of forest in a grid cell and the ability of possums to traverse such a habitat. Be aware we want to model resistance, hence the values in ts are code between zero and one and zero means (100% forest), whereas one (no trees) in a grid cells. Rivers are also coded between zero and one, meaning is a large river and towards zero a smaller tributary. Bridges are “coded” as gaps in rivers. Have a look at the resistance layers.

```
ts <- raster("./data/TreeScrub.tif")
rb <- raster("./data/RiverBridges.tif")

par(mfrow = c(2, 1))
plot(ts)
plot(rb)
```



Actually we wanted to reproject lat-lons to the coordinate of the resistance layers.

```
proj4string(rb)
```

```
## [1] "+proj=tmerc +lat_0=0 +lon_0=173 +k=0.9996 +x_0=1600000 +y_0=10000000 +ellps=GRS80 +units=m +no_defs"
```

```
proj4string(ts)
```

```
## [1] "+proj=tmerc +lat_0=0 +lon_0=173 +k=0.9996 +x_0=1600000 +y_0=10000000 +ellps=GRS80 +units=m +no_defs"
```

Both are in a coordinate system commonly used in NZ: NZGD2000 (EPSG: 2193). We therefore need to project our NZ possum data into this system and put that information in the @other\$xy slot, where it can be processed by our functions.

```
library(rgdal)
```

```
## rgdal: version: 1.4-3, (SVN revision 828)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 2.2.3, released 2017/11/20
## Path to GDAL shared files: C:/Program Files/R/library/rgdal/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
## Path to PROJ.4 shared files: C:/Program Files/R/library/rgdal/proj
## Linking to sp version: 1.3-1
```

```
nz.pr4 <- proj4string(rb)
```

```
xy <- project(as.matrix(poss.nz@other$latlong[,
  2:1]), proj = nz.pr4)
```

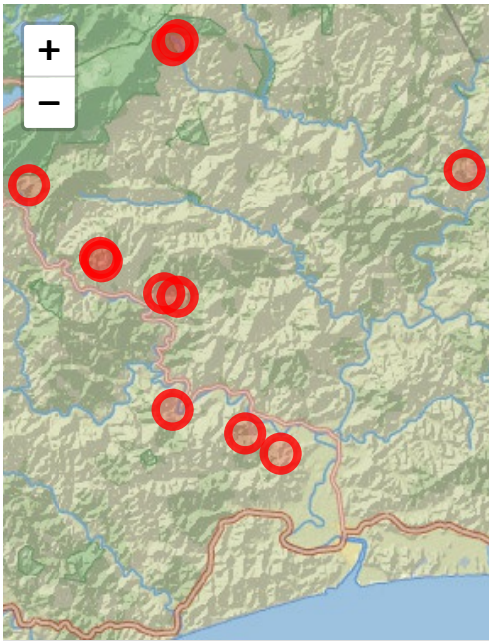
```
poss.nz@other$xy <- data.frame(x = xy[, 1], y = xy[,
  2])
```

As mentioned above we will further reduce our data set to a smaller subsample because of run-time reasons:

```
# only population 3
poss.sub.nz <- poss.nz[pop(poss.nz) == 3, ]

set.seed(5) #leave it during the first run
# only 11 individuals from that population
poss.sub.nz <- poss.sub.nz[sample(1:nInd(poss.sub.nz),
  11, replace = F), 1:1000]

gl.map.interactive(poss.sub.nz)
```



Leaflet | © OpenStreetMap contributors, CC-BY-SA, Tiles © Esri — National Geographic, Esri, DeLorme, NAVTEQ, UNEP-WCMC, USGS, NASA, ESA, METI, NRCAN, GEBCO, NOAA, IPC

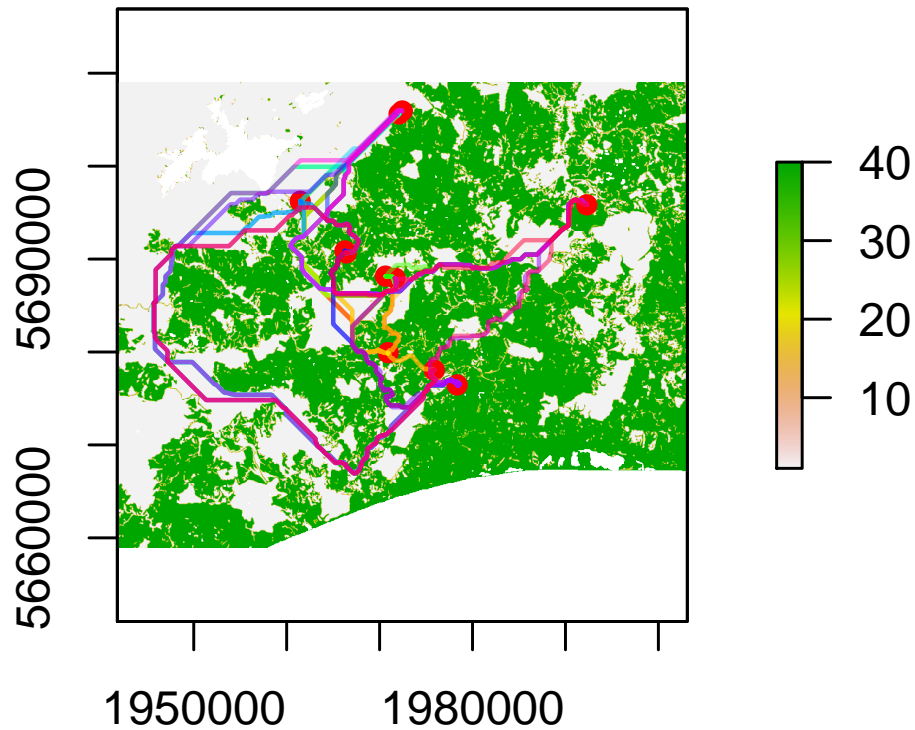
The next step is to prepare our resistance layers. We first will test only one resistance layer (ts) and its effect on the genetic distance of our 11 individuals. Currently the resistance values of ts are coded to be between zero and one (zero=100% forest one=0%forest). We assume a fairly strong linear effect of the %of non-forest and rescale the effect to be between 1 (no detour caused) for 100% forest and 40 units caused by 0% forest.

```
ts40 <- ts
values(ts40) <- (values(ts40)) * 39 + 1
```

To run a resistance analysis using an individual based distance (kosman) and least-cost paths, we type:

```
system.time(glc <- gl.genleastcost(possib.nz,
  fric.raster = ts40, gen.distance = "Kosman",
  NN = 8, pathtype = "leastcost"))
```

TreeScrub:leastcost, NN=8



```
##
|
|===          | 9%
|
|=====      | 18%
|
|=====      | 27%
|
|=====      | 36%
```

```

|
|=====| 45%
|
|=====| 55%
|
|=====| 64%
|
|=====| 73%
|
|=====| 82%
|
|=====| 91%
|
|=====| 100%

```

```

##      user  system elapsed
##    62.72   18.75   81.56

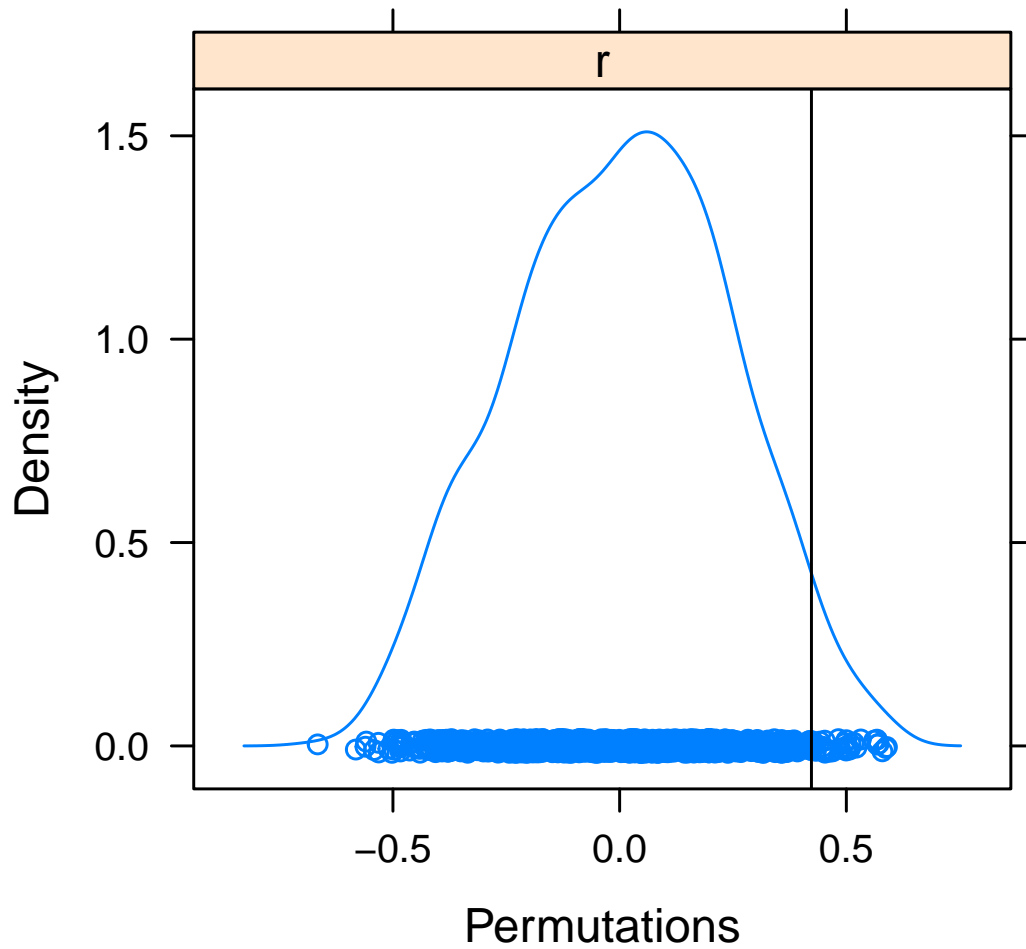
```

```

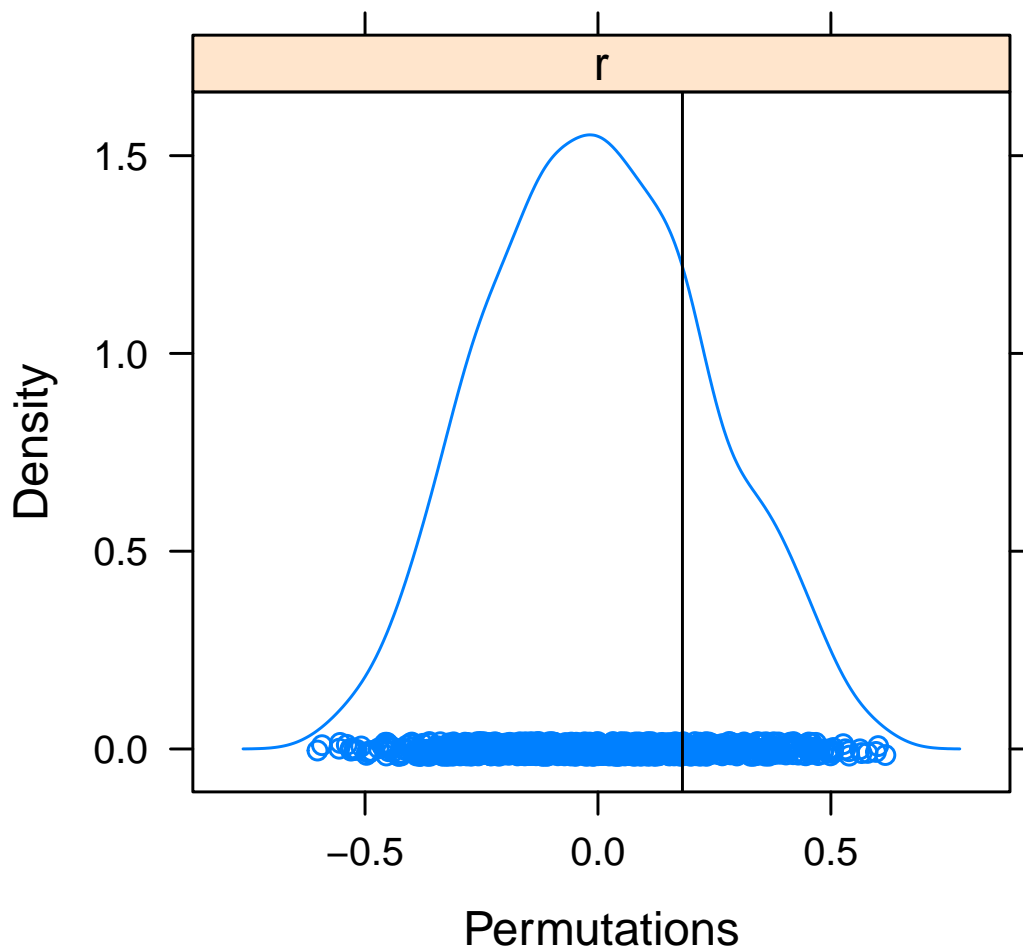
wassermann(eucl.mat = glc$eucl.mat, cost.mat = glc$cost.mats,
  gen.mat = glc$gen.mat)

```

Gen ~TreeScrub | Euclidean



Gen ~Euclidean | TreeScrub



```
## $mantel.tab
##               model      r      p
## 1 Gen ~TreeScrub | Euclidean 0.4231 0.035
## 2 Gen ~Euclidean | TreeScrub 0.1814 0.22
```



Question

Do you find a “significant” effect of the amount of forest on the population structure of possums in NZ?



Task

The next part is a practice run for you. You will rerun the analysis and interpret its result using a raster stack (several resistance layers in one go).

We create 3 layers, the one we just used (ts40) another one that checks for an effect of rivers (rb50) and then one layer that combines both effects.

1. What is your finding?
2. Actually it is not a very good idea to use these three layers. Why?

The code below shows you how to combine layers in a raster stack and how to run

```
#river/bridge resistance
rb50 <- rb
values(rb50) <- values(rb50)*200+1

#combine rivers and forest
tcrb <- ts
values(tcrb) <- (values(ts))*39 + values(rb)*99 + 1
mats <- stack(ts40, rb50, tcrb)

names(mats)<- c("ts40", "rb50", "tsrb")

system.time(glc <- gl.genleasteastcost(possib.nz, fric.raster =mats, gen.distance = "Kosman", NN = 8, pathtype =

lgrMMRR(gen.mat = glc$gen.mat, cost.mats = glc$cost.mats, eucl.mat = glc$eucl.mat
```



Hint

If you could not answer question 2 above, check the “correlation” between the resistance layers.

```
library(GGally)
ss <- sapply(glc$cost.mats, function(x) as.dist(x))
ggpairs(data.frame(ss))
```

III. A simulated resistance surface landscape analysis

An excellent way to test an approach and understand its limitation, is to use a simulation approach. You will be provided with functions to:

A. simulate a landscape (a resistance layer) B. Populate this landscape with populations of a simulated species C. “Run” the species over the landscape for a number of generations and perform a landscape genetic analysis using a resistance layer.

Then you can go back and explore the parameters and its effect on the performance of the approach (e.g. change the number of loci, change the number of generations etc.)

We need to install another package: `secr`.

```
install.packages("secr")
library(secr)
library(PopGenReport) #should already be installed
library(raster) #should already be installed
```

Before we can start we also need to download some additional functions that simplifies the coding and scripts.

```
source("https://raw.githubusercontent.com/green-striped-gecko/lgfun/master/lgfun.r")
```

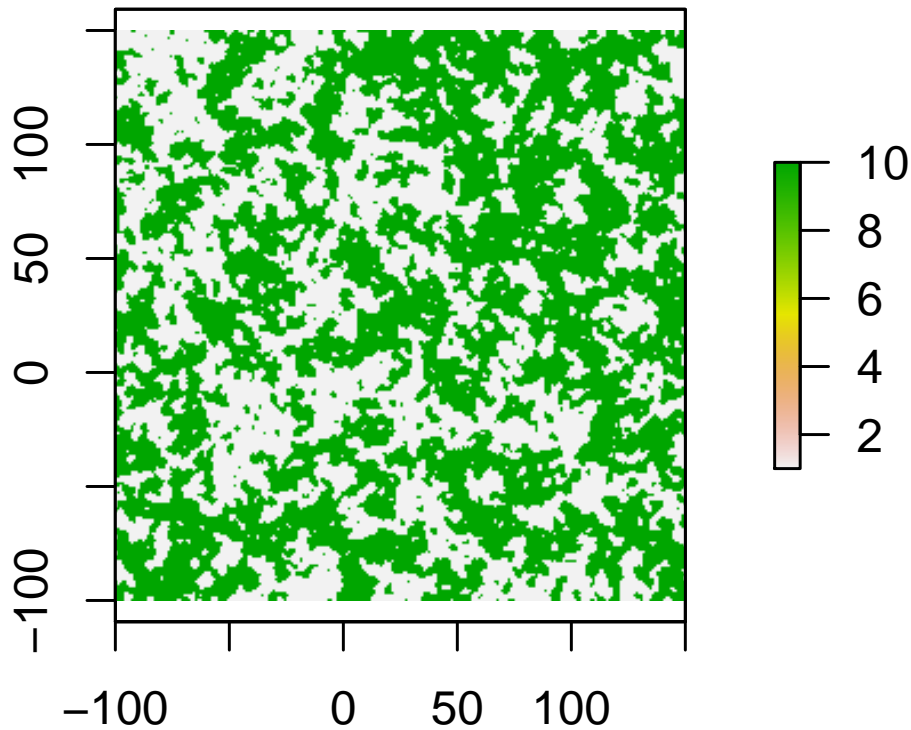
A. Simulate a resistance layer

We will use the ‘randomHabitat’ function from the ‘secr’ package to create a resistance layer, but you could simply load a png file or any other image file format with the ‘raster’ function from package ‘raster’ (?raster::raster, see the examples in there).

Parameter A is the amount of habitat and p controls the amount of clumping. Nx, ny determine the extend of the landscape and resVal determines the resistance values of non-Habitat features.¹

¹ Check ?randomHabitat for details

```
r <- create.resistance(nx = 50, ny = 50, p = 0.5,
  A = 0.5, resVal = 10)
```



Task

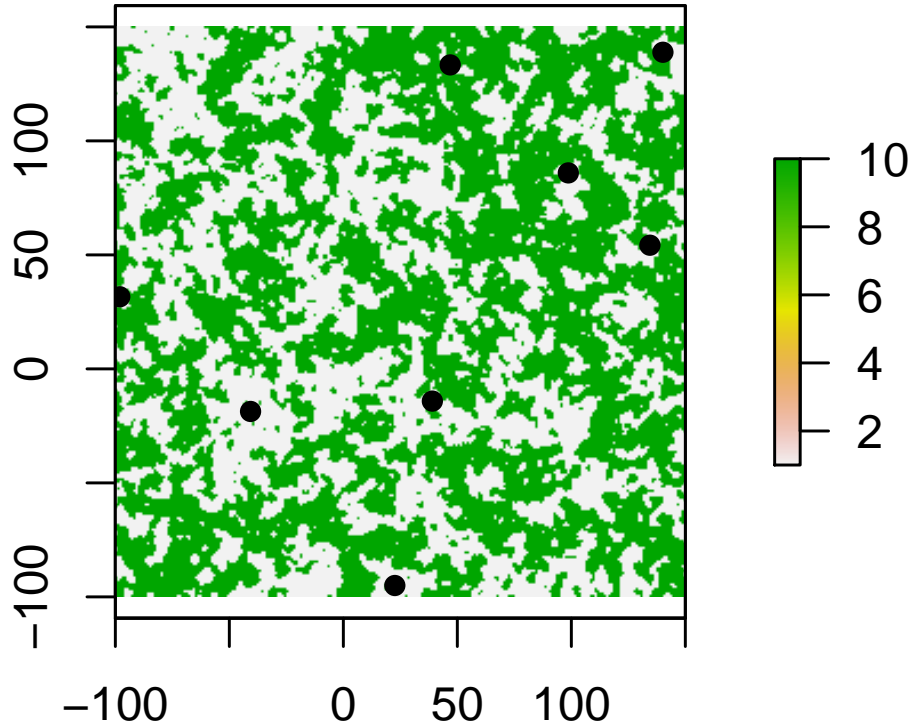
1. Rerun the command above until you “like” the resistance layer. You can change the parameter values (but be aware that p should not be higher than 0.55, otherwise the algorithm does not work reliably).

B. Populate the landscape with population of a species

B.1. Add populations to the landscape (using minimal distance)

`createpops` allows us to set up ‘n’ subpopulations in the habitat only (non green areas). The subpopulations should be at least ‘mindist’ units apart.

```
locs <- create.pops(n = 8, mindist = 3, landscape = r,
  plot = TRUE)
```



```
locs
```

```
##      lx      ly
## 1 134.47039 54.24587
## 2  46.86985 133.33752
## 3 -40.68801 -18.66652
## 4  22.58607 -95.00428
## 5  39.03843 -14.13153
## 6  98.68048  85.96216
## 7 -98.07192  31.64083
```

```
## 8 140.16925 138.87580
```

B.2. Initialise a metapopulation

We use 'init.poggensim' from package PopGenReport to initialise a metapopulation based on the locations we created earlier. To do this we need to initialise a number of parameters (the locations of the subpopulations, the number of individuals per subpopulation, the number of loci and alleles per loci. For a full list check '?init.poggensim').

To store all the parameters we create a list called para where we store all of them

Define metapopulation:

```
para <- list()
# Define populations (dynamics)
para$n.pops = 8
para$n.ind = 50

para$sex.ratio <- 0.5
# age distribution....

para$n.cov <- 3
# number of covariates (before the loci in the
# data.frame, do not change this!!)
```

Define population dynamics:

```
# reproduction
para$n.offspring = 2

# migration
para$mig.rate <- 0.2

# dispersal: exponential dispersal with
# maximal distance in map units
para$disp.max = 100 #average dispersal of an individual in meters
para$disp.rate = 0.1 #proportion of dispersing individuals

# Define genetics [create 100 SNPs]
para$n.allels <- 2
para$n.loci <- 200
para$mut.rate <- 1e-05
```

Define a cost distance method:

```
para$method <- "leastcost" #rSPDdistance, commute
para$NN <- 8 #number of neighbours for the cost distance method
```

```

# Initialize simulation of populations from
# scratch

landscape <- r #<-raster(system.file('external/rlogo.grd', package='raster'))

# Define x and y locations
para$locs <- locs
# give the population some names
rownames(para$locs) <- LETTERS[1:para$n.pops]

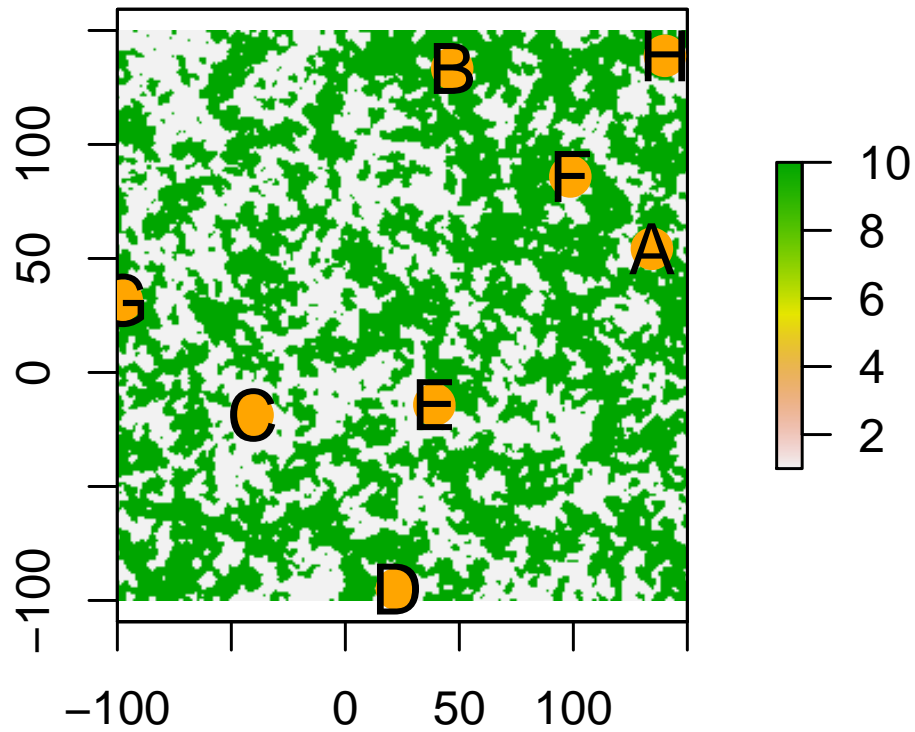
# Create a costdistance matrix

cost.mat <- gl.costdistances(landscape, para$locs,
                             para$method, para$NN)
# needed for the simulation
eucl.mat <- as.matrix(dist(para$locs)) #needed for the analysis later

# Plot your landscape with the populations....

plot(landscape)
points(para$locs[, 1], para$locs[, 2], pch = 16,
       cex = 2, col = "orange")
text(para$locs[, 1], para$locs[, 2], row.names(para$locs),
     cex = 1.5)

```



```
# Check the parameter list
```

```
para
```

```
## $n.pops
## [1] 8
##
## $n.ind
## [1] 50
##
## $sex.ratio
## [1] 0.5
##
```

```
## $n.cov
## [1] 3
##
## $n.offspring
## [1] 2
##
## $mig.rate
## [1] 0.2
##
## $disp.max
## [1] 100
##
## $disp.rate
## [1] 0.1
##
## $n.allels
## [1] 2
##
## $n.loci
## [1] 200
##
## $mut.rate
## [1] 1e-05
##
## $method
## [1] "leastcost"
##
## $NN
## [1] 8
##
## $locs
##      lx      ly
## A 134.47039 54.24587
## B  46.86985 133.33752
## C -40.68801 -18.66652
## D  22.58607 -95.00428
## E  39.03843 -14.13153
## F  98.68048  85.96216
## G -98.07192  31.64083
## H 140.16925 138.87580
```

B.3 Initialise your population on the landscape

Now finally we can initialise our population using the `init` function


```
simpops <- init.popgensim(para$n.pops, para$n.ind,
  para$sex.ratio, para$n.loci, para$n.allels,
  para$locs, para$n.cov)
```

You may want to check the simpops object, which is simply a list of our subpopulation and each individual is coded in a single run in one of the subpopulations.

```
names(simpops) #the names of the subpopulations
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H"
```

We can also analyse our simpop object. (e.g. calculate the pairwise Fst value between all the populations).

To be able to do that we first need to convert it into a genlight object, then we can use our usual functions (e.g gl.fst.pop).

```
glsp <- pops2gl(simpops, locs = para$locs)
```

```
## Loading required package: parallel
```

```
glsp #check the genlight object
```

```
## /// GENLIGHT OBJECT //////////
##
## // 400 genotypes, 200 binary SNPs, size: 648.3 Kb
## 0 (0 %) missing data
##
## // Basic content
## @gen: list of 400 SNPbin
## @ploidy: ploidy of each individual (range: 2-2)
##
## // Optional content
## @ind.names: 400 individual labels
## @loc.names: 200 locus labels
## @pop: population of each individual (group size range: 50-50)
## @other: a list containing: xy
```

```
summary(glsp)
```

```
## Length Class Mode
##      1 genlight S4
```

```
gen.mat <- as.matrix(as.dist(gl.fst.pop(glsp,
  nboots = 1)))
# overall pairwise fst
mean(as.dist(gen.mat))
```

```
## [1] -6.659687e-05
```



Task

1. Check the pairwise F_{st} s between populations. Why are they so low? Is there an effect of the landscape on the population structure, yet?
2. Run a partial mantel test and check the result []

Now we run our simulation by simply passing our `simpops`, with some additional parameters that are needed for the simulation. The number of generations the simulation should run is in the `steps` parameter. (check `?run.popgensim` for a description of all parameters).

Important to understand is the idea of the `cost.mat` (which is the cost matrix that is used for the distance between subpopulation).

C. Run simulations and analyze results**

C.1. Run your population several generations (steps) on the landscape

```
simpops <- run.popgensim(simpops, steps = 3, cost.mat,
  n.offspring = para$n.offspring, n.ind = para$n.ind,
  para$mig.rate, para$disp.max, para$disp.rate,
  para$n.allels, para$mut.rate, n.cov = para$n.cov,
  rec = "none")
```

In essence we were running a metapopulation with 100 individuals per subpopulation on our resistance landscape for 3 generations. The question is now, was that enough time to create an effect on the population structure?

We can check now the pairwise F_{st} values and then do a landscape genetic analysis using partial mantel tests.



Task

1. Check the pairwise Fsts between populations after 3 generation have passed. Is there now an effect of the landscape on the population structure, yet?
2. Run a partial mantel test and check the result
3. Run now your population another 50 generation.

```
simpops <- run.popgensim(simpops, steps=50,...)
```

How did the overall mean pairwise Fst change?

Is there a significant effect of the landscape on the population structure after 53 generations?

Example how to use the simulator: We are interested how many generation it takes until the average Fst values becomes so “different”, that the effect of the resistance approach can be detected. We create a loop and calculate mean pairwise Fst values every second generation (up to 20). We plot generations against average pairwise Fst and colour the dots by the significance of the Mantel test.

```
# initialise
simpops <- init.popgensim(para$n.pops, para$n.ind,
  para$sex.ratio, para$n.loci, para$n.allels,
  para$locs, para$n.cov)
# Calculate overall Fsts
glsp <- pops2gl(simpops, locs = para$locs)
gen.mat <- as.matrix(as.dist(gl.fst.pop(glsp,
  nboots = 1)))
# overall pairwise fst
tempfst <- mean(as.dist(gen.mat))

glc <- gl.genleastcost(glsp, fric.raster = landscape,
  gen.distance = "D", NN = 8, pathtype = "leastcost")
temp.mantel <- wassermann(gen.mat = glc$gen.mat,
  eucl.mat = glc$eucl.mat, cost.mats = glc$cost.mats,
  plot = F)

# find the p value in temp
mantelp <- as.numeric(temp.mantel$mantel.tab["1",
  "p"])

res <- data.frame(generation = 0, fst = tempfst,
  mantelp)
```

```

for (i in 1:10) {
  simpops <- run.popgensim(simpops, steps = 2,
    cost.mat, n.offspring = para$n.offspring,
    n.ind = para$n.ind, para$mig.rate, para$disp.max,
    para$disp.rate, para$n.allels, para$mut.rate,
    n.cov = para$n.cov, rec = "none")

  glsp <- pops2gl(simpops)
  gensim.mat <- as.matrix(as.dist(gl.fst.pop(glsp,
    nboots = 1)))
  # overall pairwise fst
  tempfst <- mean(as.dist(gensim.mat))
  temp.mantel <- wassermann(gen.mat = gensim.mat,
    eucl.mat = glc$eucl.mat, cost.mats = glc$cost.mats,
    plot = F)

  mantelp <- as.numeric(temp.mantel$mantel.tab["1",
    "p"])

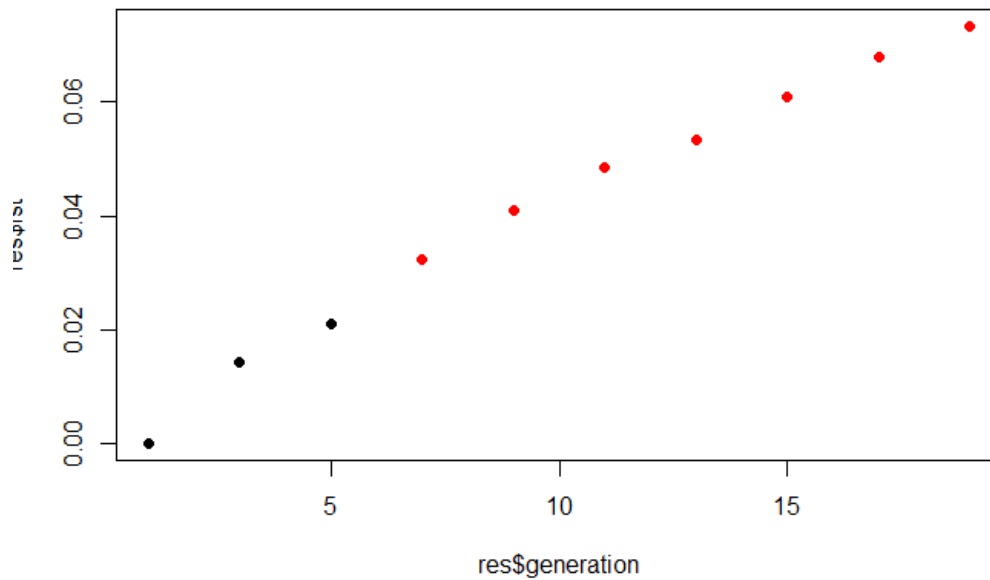
  res[i, ] <- c(i * 2, tempfst, mantelp)

  cat(paste("Generation:", i * 2, "\n"))
}

plot(res$generation, res$fst, col = (res$mantelp <
  0.05) + 1, pch = 16)

res

```



Task

You can now “play” with the simulator using different landscapes, number of subpopulations, different locations, number of loci etc. For example rerun your analysis for only 4 subpopulations. How does this affect your ability to detect an effect of the landscape? Or you can try to use your own data and create a resistance layer. Be aware that the coordinate system between your landscape and genlight object needs to be the same preferably projected coordinate system.

IV. Create a resistance layer for your own data (a bit experimental!!!)

Admittedly this is a bit of a stretch, as originally you would like to have a gis data set with landscape features to create a resistance layer. Here will use a “quick and dirty” approach and create a resistance landscape from open source maps, which may be not too useful for your data set.

The idea of this tutorial is to demonstrate the approach for an example of our fox data
Make sure you have installed two additional libraries:

```
# install via: library(devtools)
# install_github('dkahle/ggmap')
library(ggmap)
library(dartR)
library(osmdata)
# also load some helper functions

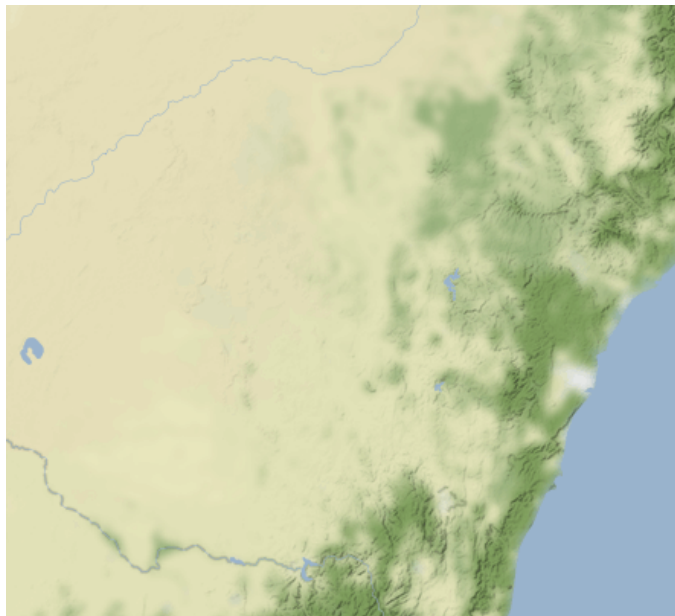
source("https://raw.githubusercontent.com/green-striped-gecko/dartRworkshop/master/code/lgfuns.r")

## define your genlight data set here we use
## the bandicoots in NSW
glNSW <- bandicoot.gl[pop(bandicoot.gl) == "NSW",
  1:100]

## bounding box in lat/lon
ll <- glNSW@other$latlong
bb <- matrix(c(min(ll$lon), min(ll$lat), max(ll$lon),
  max(ll$lat)), nrow = 2, ncol = 2)

# or you can use geocode functions bb <-
# getbb('South Australia')

# get a map you may need to play with zoom to
# get a suitable map also you could try
# terrain-lines to get roads if your study is
# close to a city
map <- get_stamenmap(bb, zoom = 6, maptype = "terrain-background")
plot(map)
```



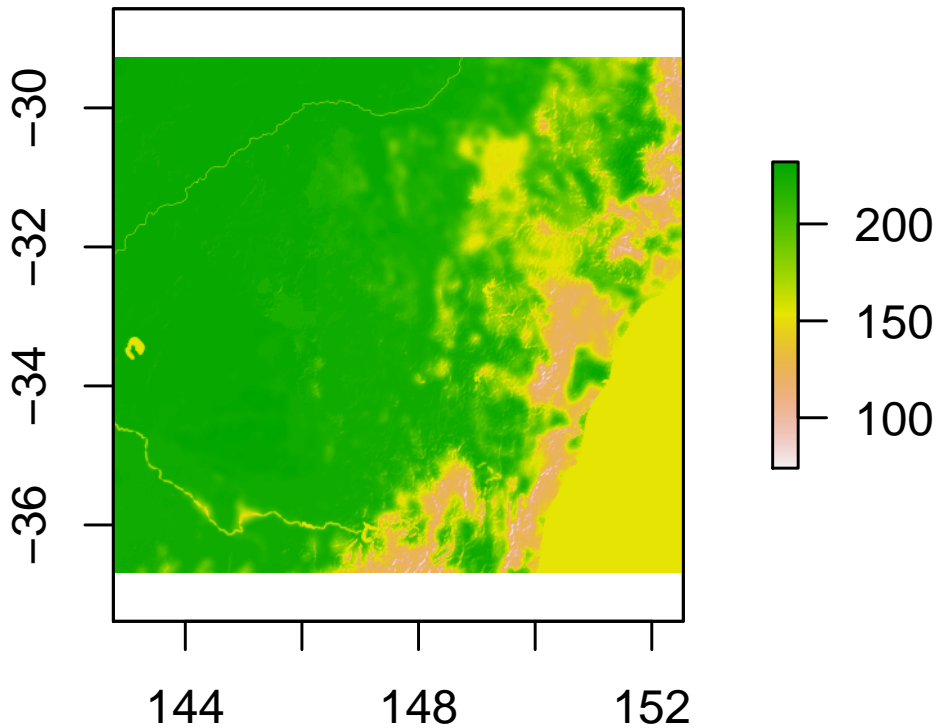
We then need to convert the image to a raster

```
## converts the image to a raster (only the red
## values are returned)
map.raster <- ggmap_to_raster(map)[[1]]
map.raster

## class      : RasterLayer
## dimensions  : 402, 444, 178488 (nrow, ncol, ncell)
## resolution  : 0.02201689, 0.01840868 (x, y)
## extent     : 142.7657, 152.5412, -36.6801, -29.27981 (xmin, xmax, ymin, ymax)
## coord. ref. : NA
## data source : in memory
## names      : layer.1
```

```
## values      : 74, 232 (min, max)
```

```
plot(map.raster)
```



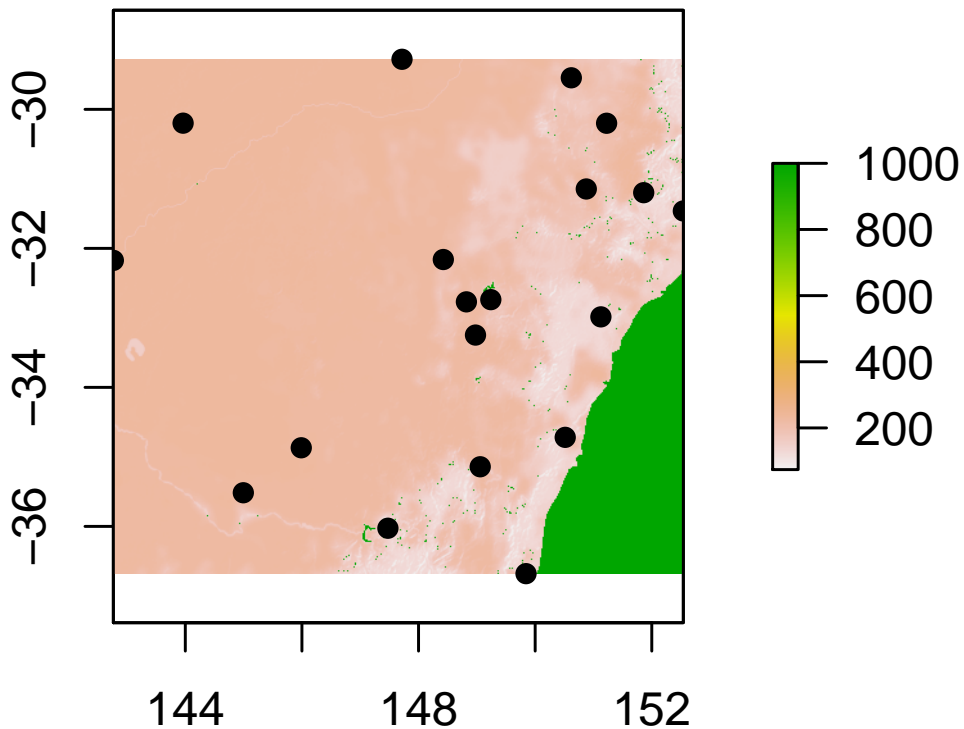
Obviously this is not a very suitable map, basically colors (the amount of red of each pixel is converted into resistance), nevertheless for demonstration purposes we will use it.

To make it a bit more realistic we want to convert the ocean to NAs. As we can see the ocean has a “greenness” that refers to a value around 150. If you table the values of the raster you see, that values of 153 are very prominent. To make them “unavailable” we set those to very high resistance values=1000 (to make sure paths are not running through the ocean), we use:

```
reslay <- map.raster #copy to keep the original map
reslay[values(reslay) == 153] <- 1000
plot(reslay)
```



```
points(glNSW$other$latlong, pch = 16)
```



Now we can run a resistance layer analysis. The difference this time is that we run it on individuals, hence we need to specify an individual distance measure (the fastest in terms of computer time is the proportion of Shared alleles [=propShared]).

Two more issues:

- If you inspect the points in the map above closely you see that four individuals are outside/at the boundary of the actual resistance layer. We would like to delete them.

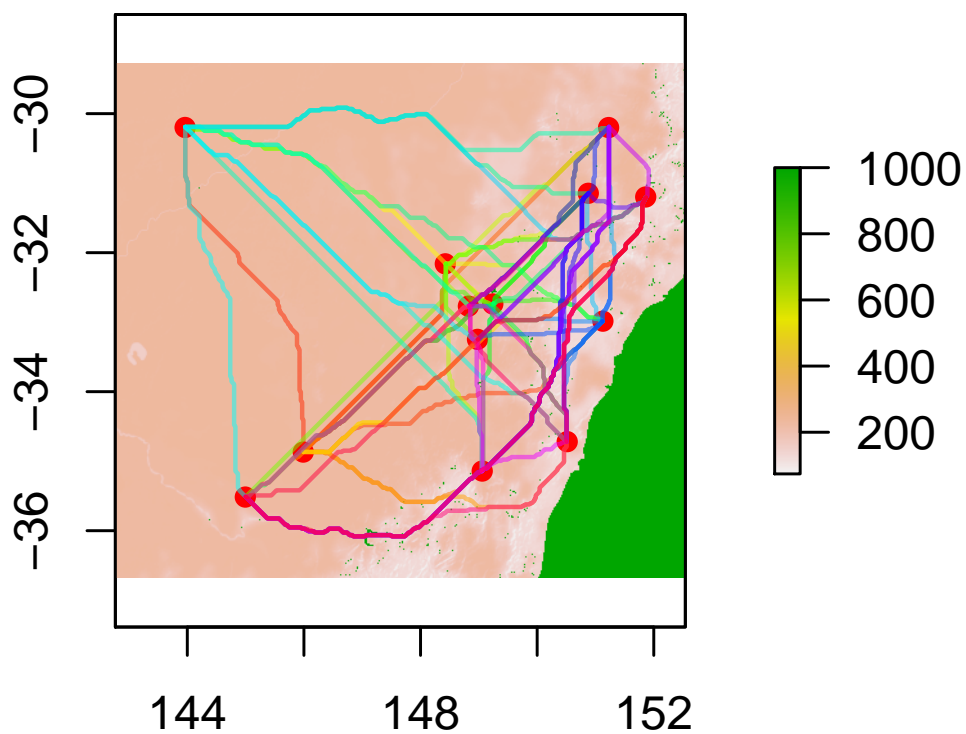
```
glNSW2 <- glNSW[glNSW$other$latlong$lon < 152 &  
  glNSW$other$latlong$lon > 143 & glNSW$other$latlong$lat <  
  -30 & glNSW$other$latlong$lat > -36, ]
```

- The other issue is that we would need to reproject the data set so coordinates are in suitable non-geographic units. For simplicity we skip this step here and will receive a warning.

```
glc <- gl.genleastcost(glNSW2, fric.raster = reslay,
  gen.distance = "propShared", NN = 8, pathtype = "leastcost")
```

```
## No projected coordinates in @other$xy found. Hence will use latlongs (if provided), which are not project
```

layer.1:leastcost, NN=8



```
##
|
|===                | 8%
|
|=====            | 15%
```

```

|
|=====| 23%
|
|=====| 31%
|
|=====| 38%
|
|=====| 46%
|
|=====| 54%
|
|=====| 62%
|
|=====| 69%
|
|=====| 77%
|
|=====| 85%
|
|=====| 92%
|
|=====| 100%

```

You can see the warning about the projection and yes I would encourage to project your data set to a suitable coordinate system.

The final step is now to run a partial mantel test on the matrices. Same code as before:

```
wassermann(gen.mat = glc$gen.mat, eucl.mat = glc$eucl.mat,
  cost.mats = glc$cost.mats, plot = F)
```

```
## $mantel.tab
##              model      r      p
## 2 Gen ~Euclidean | layer.1 0.0126 0.464
## 1 Gen ~layer.1 | Euclidean -0.0123 0.549
```

Not surprisingly there is not effect of the resistance layer on the genetic structure (nor an isolation by distance effect).



Task

Now you should have all the ingredients to create your own resistance layer and to run your own data set.



Hint

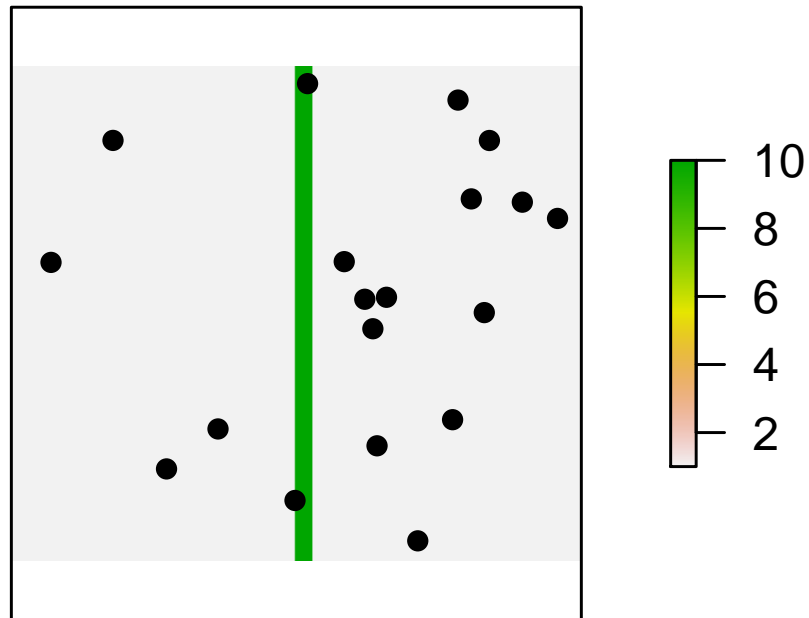
Another, way to create a resistance layer is to “draw” it in R. Here you basically create a raster with the correct extend and then draw lines on top of it.

Or you simply draw it in your favourite image editor and load it into R with the **raster** function. Be aware here you would need to set the extent manually to match your coordinates.

```
r <- raster(xmn = 142, xmx = 153, ymn = -37, ymx = -29)
r[] <- 1
r
```

```
## class      : RasterLayer
## dimensions  : 180, 360, 64800  (nrow, ncol, ncell)
## resolution  : 0.03055556, 0.04444444  (x, y)
## extent     : 142, 153, -37, -29  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 1  (min, max)
```

```
rbarrier <- r
rbarrier[, 180:190] <- 10
plot(rbarrier, axes = FALSE)
points(glNSW@other$latlong, pch = 16)
```



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
```



Well done!!

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

Cheers, Bernd