Landscape Genetics Intro

Introduction to landgenreport

Bernd Gruber & Aaron Adamack

October 16, 2014

Contents

1 Preface **3**

2 A simple least-cost path modelling analysis (LCPMA) **3**
- 2.1 Genetic data set - create a genind object [with coordinates] 4
  - 2.1.1 Add spatial information to a genind object 7
- 2.2 Creating a cost matrix layer 9
- 2.3 Selecting a relatedness index 10
- 2.4 Run the created example 11

3 The results of a least-cost path modelling analysis **11**
- 3.1 A map of the least-cost paths 12
- 3.2 Checking the results of partial mantel tests and MMRR 14
- 3.3 Output of the first example 15
- 3.4 Use the read.genetable function to create a genind object with coordinates in one go 17

4 How to project latlong data into Mercator **19**

5 A customised step by step analysis **23**

6 Contacts and Citation **35**

7 References **36**

8 Appendix **36**

1 Preface

This tutorial provides an overview how to use landgenreport to perform a least-cost path modeling analysis as a part of a landscape genetic analysis. The tutorial assumes that you already have a basic understanding of how to use R. For example you should know how to run a function and load a package. If you don't have this level of understanding of R, there are a number of excellent tutorials and introductions to R available online. You can find them by using your favourite search engine to search for "R introduction" or "R tutorial". To start using this tutorial, you will need to install the package PopGenReport and (optionally) have a LaTeX environment installed on your computer in order to make full use of the package. For information on how to install PopGenReport and LaTeX please visit our website: `http://www.popgenreport.org` and/or look at the other PopGenReport package tutorial (PopGenReportIntroduction) [via `browseVignettes("PopGenReport")`]. If you are using the Windows operating system, you can alternatively download a mobile version of the package (including a running LaTeX environment): PopGenPack. This allows you to either run the package from a USB device or copy it to a specified folder on your system, without the need to setup all of the software packages. Finally, you should read a publication on least-cost path modelling [e.g. Gruber & Adamack (in prep.)] to understand the principles behind the least-cost path modelling approach.

2 A simple least-cost path modelling analysis (LCPMA)

After starting R with your favourite R Editor (we suggest Rstudio [`http://www.rstudio.org`] you will need to load the latest version of the package PopGenReport into your workspace and confirm that you are running at least Version 2.0, by typing:

```
require(PopGenReport)
#check which version you have
packageVersion("PopGenReport")

## [1] '2.1'

#If you have a version before 2.0 then you need to update the package
#using the command install.packages (remove the hash symbol in the next line)
#install.packages("PopGenReport")
```

Once you have confirmed that you have the right version of PopGenReport, you can run a simple LCPMA by typing the command shown in the example text below. Please note (this is true for every command line using the function `landgenreport`):
You need to change the argument mk.pdf=FALSE to mk.pdf=TRUE if you want a full report. The argument is set to FALSE here, to make sure that the examples run even if you dońt have a LaTeX environment installed, which is not recommended, but may be the case for users who do not want to install LaTeX or do not have the necessary admin rights on their computer.

```
#remember for a full report set: mk.pdf=TRUE
results1 <-landgenreport(landgen, fric.raster, "D", NN=4,  mk.pdf=FALSE)

## Loci names were not unique and therefore adjusted.
## Compiling report...
## - Landscape genetic analysis using resistance matrices...
```

```
## Analysing data ...
## All files are available in the folder:
## C:\Users\s425824\AppData\Local\Temp\RtmpOU2oaT/results

#let us check if there is anything returned to our results1 object
names(results1)

## [1] "leastcost"

#there should be a list called: leastcost in your results1 object
```

The complete output of our least-cost path modelling analysis (LCPMA) is stored in the results folder at the indicated path (the exact location depends on the temporary folder on your machine). In this folder you should find a number of CSV, PDF, SVG and other files. Please see if you can locate the files. If you set mk.pdf=TRUE you will find a full report of your analysis in a single PDF named LandGenReport.pdf (this PDF can be found at the end of this tutorial in the section 8: Appendix).

Hint: You can set the output folder to a more convenient location by setting the output path via the `path.pgr` argument. Please make sure the specified folder does exist, for example:

```
results<-landgenreport(landgen,fric.raster,"D",NN=4,path.pgr="D:/temp")
```

There are three essential bits of information that needs to be provided to the `landgenreport` function:

1. landgen: a genetic data set that contains the genotypes and coordinates of several individuals (a genind object)
2. fric.raster: the cost layer of the landscape (a raster object)
3. "D": a character which indicates the type of genetic distance index that will be used to create the pairwise-genetic distance matrix, e.g Jost's D in this case

For the example above, all three types of information have been provided in the correct format. Now we will run a real world example, that details the steps necessary to create the three objects for your own data set.

2.1 Genetic data set - create a genind object [with coordinates]

We start with the genetic data set. Quite often, you will already have your data formatted for another genetic analysis program such as: STRUCTURE, GENETIX, GENEPOP or FSTAT. These file formats can easily be converted into a `genind` object. Below, we provide some example for converting your data into a genind object if your data is already in the format for STRUCTURE.

First you need to tell R where your data set is located. Please be very exact when you provide the path to your file, otherwise R will not be able to find it. For this example, we will assume your data is stored in the folder 'D:/data' and is named 'testdata.struc'. If your data is in the STRUCTURE format, simply type:

```
gendata <- read.structure('D:/data/testdata.struc')
```

For other data formats, the command is very similar:

```
gendata <- read.genetic('D:/data/testdata.gtx') # for genetix files
gendata <- read.genepop('D:/data/testdata.gen') # for genepop files
gendata <- read.fstat('D:/data/testdata.dat') # for fstat files
```

For the next example I use a structure file that is provided by the package `adegenet` called nancycats. After the file name you need to provide some details on how your STRUCTURE file is formatted. For more details, just type the function name preceded by a "?". For `read.structure` you need to type `?read.structure` to get to the R help pages for this function and `?nancycats` to get an overview on the data set.

```
fname <- system.file("files/nancycats.str",package="adegenet")
fname #location of the structure file

## [1] "C:/Program Files/R/library/adegenet/files/nancycats.str"

gendata <-read.structure(fname,onerowperind=FALSE, n.ind=237,
          n.loc=9, col.lab=1, col.pop=2, ask=FALSE)

##
##  Converting data from a STRUCTURE .stru file to a genind object...
```

Using the read functions above your data will be converted into a genind object. You can confirm that your dataset is now a genind object by typing:

```
class(gendata)

## [1] "genind"
## attr(,"package")
## [1] "adegenet"
```

We can explore the content of the genind object by typing its name "gendata" into the console.

```
gendata

##
##    ######################
##    ### Genind object ###
##    ######################
## - genotypes of individuals -
##
## S4 class:  genind
## @call: read.structure(file = fname, n.ind = 237, n.loc = 9, onerowperind = FALSE,
##     col.lab = 1, col.pop = 2, ask = FALSE)
##
## @tab:  237 x 108 matrix of genotypes
##
## @ind.names: vector of  237 individual names
## @loc.names: vector of  9 locus names
## @loc.nall: number of alleles per locus
## @loc.fac: locus factor for the  108 columns of @tab
## @all.names: list of  9 components yielding allele names for each locus
```

```
## @ploidy:  2
## @type:  codom
##
## Optional contents:
## @pop:  factor giving the population of each individual
## @pop.names:  factor giving the population of each individual
##
## @other: - empty -
```

The slots of the genind object (indicated by the @ sign which are subcomponents of gendata) can also be examined by typing their names into the console. For example, there are slots that indicate the population that each individual belongs to (@pop), the loci names (@loc.names) and the number of alleles per locus (@loc.nall). You should always check this information to confirm that your data has loaded correctly.

```
gendata@pop
```

```
##   [1] P01 P01 P01 P01 P01 P01 P01 P01 P01 P01 P02 P02 P02
##  [14] P02 P02 P02 P02 P02 P02 P02 P02 P02 P02 P02 P02 P02
##  [27] P02 P02 P02 P02 P02 P02 P03 P03 P03 P03 P03 P03 P03
##  [40] P03 P03 P03 P03 P03 P04 P04 P04 P04 P04 P04 P04 P04
##  [53] P04 P04 P04 P04 P04 P04 P04 P04 P04 P04 P04 P04 P04
##  [66] P04 P04 P05 P05 P05 P05 P05 P05 P05 P05 P05 P05 P05
##  [79] P05 P05 P05 P05 P06 P06 P06 P06 P06 P06 P06 P06 P06
##  [92] P06 P06 P07 P07 P07 P07 P07 P07 P07 P07 P07 P07 P07
## [105] P07 P07 P07 P08 P08 P08 P08 P08 P08 P08 P08 P08 P08
## [118] P09 P09 P09 P09 P09 P09 P09 P09 P09 P10 P10 P10 P10
## [131] P10 P10 P10 P10 P10 P10 P10 P11 P11 P11 P11 P11 P11
## [144] P11 P11 P11 P11 P11 P11 P11 P11 P11 P11 P11 P11 P11
## [157] P11 P12 P12 P12 P12 P12 P12 P12 P13 P13 P13 P13 P13
## [170] P13 P13 P13 P13 P13 P13 P13 P13 P14 P14 P14 P14 P14
## [183] P14 P14 P14 P14 P14 P14 P14 P14 P14 P14 P14 P14 P15
## [196] P15 P15 P15 P15 P15 P15 P15 P15 P15 P15 P16 P16 P16
## [209] P16 P16 P16 P16 P16 P16 P16 P16 P16 P12 P12 P12 P12
## [222] P12 P12 P12 P17 P17 P17 P17 P17 P17 P17 P17 P17 P17
## [235] P17 P17 P17
## 17 Levels: P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 ... P17
```

```
gendata@loc.names
```

```
##   L1   L2   L3   L4   L5   L6   L7   L8   L9
## "L1" "L2" "L3" "L4" "L5" "L6" "L7" "L8" "L9"
```

```
gendata@loc.nall
```

```
## L1 L2 L3 L4 L5 L6 L7 L8 L9
## 16 11 10  9 12  8 12 12 18
```

```
# or we can use the table function to get the number of individuals
# per population
table(gendata@pop)
```

```
##
## P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15
##  10  22  12  23  15  11  14  10   9  11  20  14  13  17  11
## P16 P17
##  12  13
```

From the lines @tab and @ind.names, you can see that we have 237 individual genotypes in this data set, information on nine loci for each individual (from the lines starting @loc.nall or @loc.names), and there are ten individuals in population "P01" (from the command table(gendata@pop)).

2.1.1 Add spatial information to a genind object

The genind object is missing some information that is very important for our LCPMA. It is missing the spatial coordinates for each individual genetic sample. STRUCTURE files can include spatial coordinates, but other genetic file formats often cannot. Thus, we will show you here how to add spatial coordinates to the genind object (gendata) from a separate file. First the spatial coordinates need to be loaded into the R workspace and then added to the correct slot in our genind object, namely in the slot @other$xy. There are two requirements for your spatial data:

1. Your data is in the same coordinate system as your cost matrix layer
2. The coordinates need to be in a projected coordinate system and (not longitude and latitude), so distances can be calculated directly from the coordinates

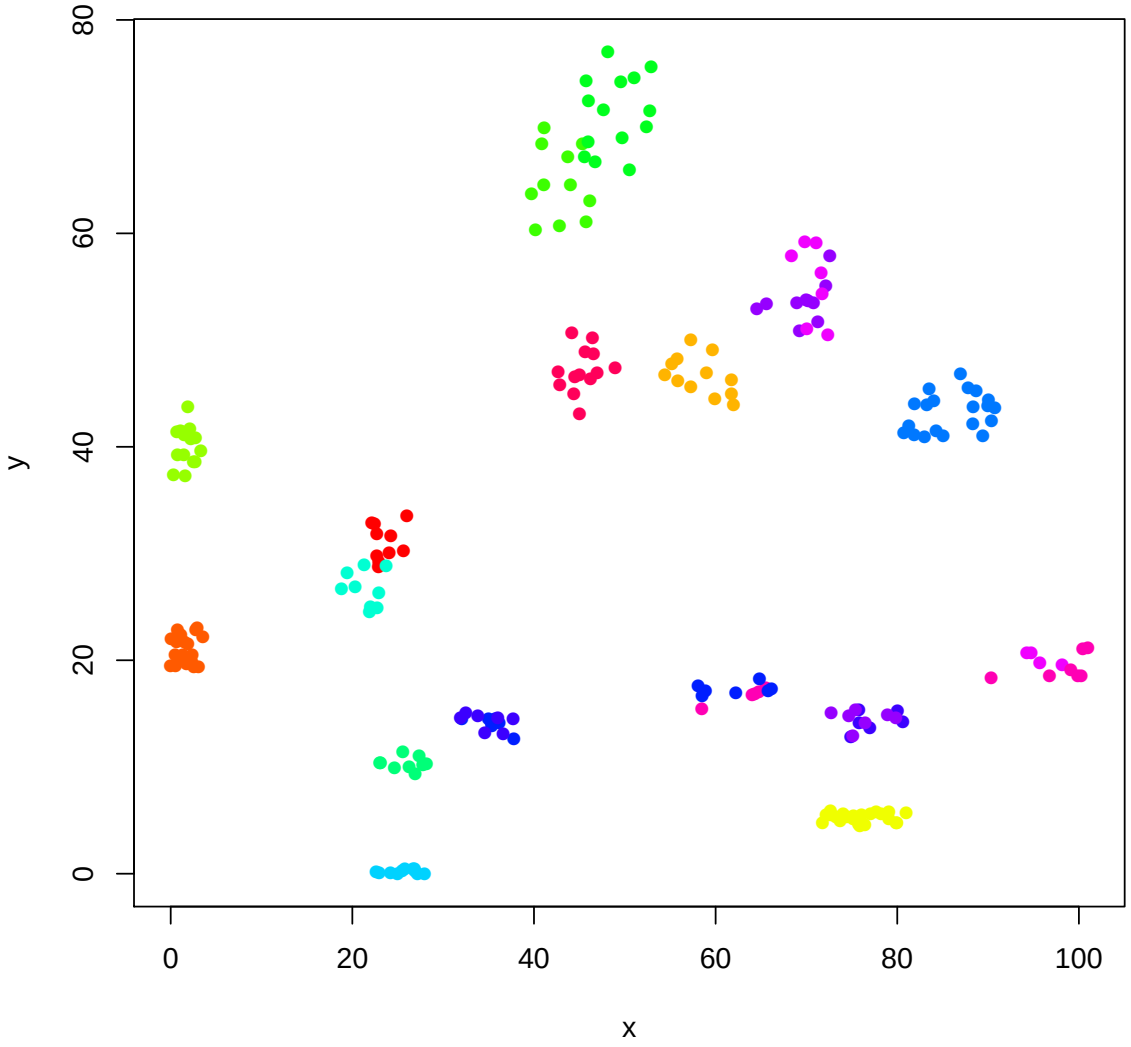
Quite often spatial coordinates are provided via latitude and longitude. If this is the case and distances are calculated without projection, this would result in imprecise estimates of spatial distances as these distance calculations would based on angles as measurement units. Therefore we demonstrate in the example below, how to project the latitude and longitude coordinates into the Mercator system (a coordinate system used by Google maps, which can be used worldwide, see Example: How to project latlong data into Mercator4)).

First we load the data from a CSV file (a comma separated text file, that for each individual has a x and a y coordinate) into the workspace using the read.csv function.

```
coordfile <- system.file("extdata/nancycoords.csv",package="PopGenReport")
coords <- read.csv(coordfile)
#we check the first 6 entries
head(coords)

##       x     y
## 1 22.71 31.84
## 2 22.89 28.73
## 3 24.24 31.62
## 4 26.02 33.56
## 5 24.06 30.07
## 6 22.43 32.79
```

We can plot the individuals by their coordinates using the plot function:

```
#define some nice colours for each population (there are 17 populations)
cols <- rainbow(17)
plot(coords, col=cols[gendata@pop], pch=16)
```



Now we can add the coordinates (coords) to our genind object (gendata). The correct place to add the coordinates is the slot @other$xy. This can be done by typing:

```
gendata@other$xy <- coords
#check if there is now a new slot @other$xy
gendata

##
##    ######################
##    ### Genind object ###
##    ######################
## - genotypes of individuals -
##
## S4 class:  genind
## @call: read.structure(file = fname, n.ind = 237, n.loc = 9, onerowperind = FALSE,
##     col.lab = 1, col.pop = 2, ask = FALSE)
##
## @tab:  237 x 108 matrix of genotypes
```

```
##
## @ind.names: vector of  237 individual names
## @loc.names: vector of  9 locus names
## @loc.nall: number of alleles per locus
## @loc.fac: locus factor for the  108 columns of @tab
## @all.names: list of  9 components yielding allele names for each locus
## @ploidy:  2
## @type:  codom
##
## Optional contents:
## @pop:  factor giving the population of each individual
## @pop.names:  factor giving the population of each individual
##
## @other: a list containing: xy
```

2.2 Creating a cost matrix layer

This part of the exercise is admittedly a bit artificial as there are numerous ways to create cost layers. The most common approach is to use a Geographic Information System (GIS) and create a raster layer that incorporates the structures which are thought to influence connectivity. For example, roads may be a barrier to dispersal and thus cells that code for roads would be given high resistance value. Alternatively, you could create an image file based on Google Earth layers (see example below), where certain landuse classes (e.g. forest) are outlined and filled with a certain colour. The colour codes for the amount of resistance value of the forest. The raster package in R is capable of loading almost every possible file formats including ESRI raster layers, ascii formats, bitmap files, geotiffs and other image formats that can be used to represent a map. All you need to provide to the raster function in R is the filepath and filename. For example if your cost matrix layer is stored using the ESRI raster file format with the name "raster.asc" in the folder "D:/data" then you simply type:

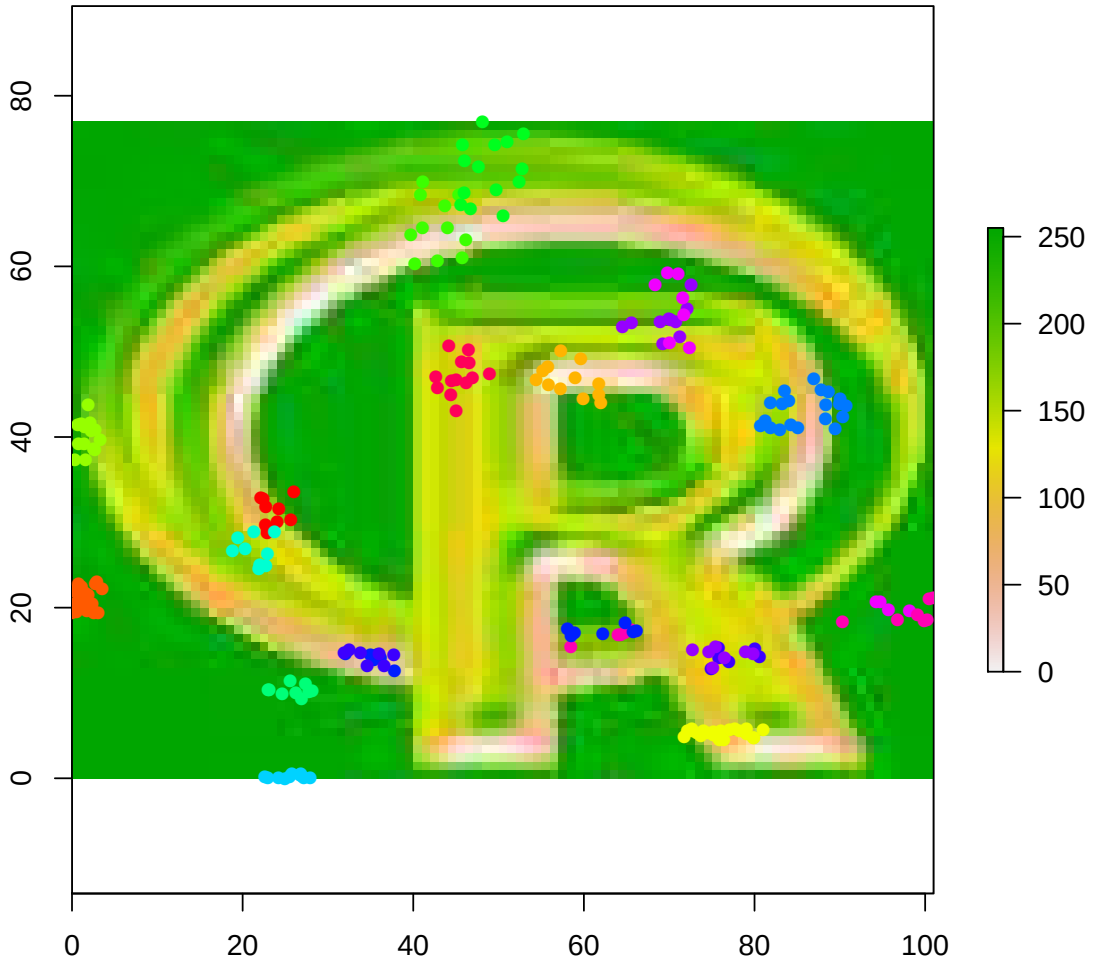
costlayer <- raster("D:/data/raster.asc")

To demonstrate that any image format can be used, we use the image of the R-logo (the red-component only) that is provided by R as our cost matrix layer. This is admittedly a useless cost matrix layer and we don't expect to find a relationship between our genetic data set and the cost matrix layer.

```
logo <- raster(system.file("external/rlogo.grd", package="raster"))
```

To test if this file has been loaded successfully, we plot it and also add the sampled individuals from our genetic data set (gendata) to the map.

```
plot(logo)
points(coords, col=cols[gendata@pop], pch=16)
```



2.3 Selecting a relatedness index

There are five indices of genetic differentation currently implemented in the `landgenreport` function (check: ?landgenreport for details). Three indices are based on a subpopulation differentiation ("D", "Gst.Hedrick", "Gst.Nei"= an often used variant of Fst) and two on relatedness between individuals ("Kosman" & Lennard, "Smouse" & Peakall). The difference between the indices is that if an individual based index is used then least-cost paths are calculated between all pairs of individuals (in the case of our example dataset there are 237*236/2 = 27966 pairs of individual genetic distances), if an index based on subpopulations is used (such as Jost's D), then the least-cost paths are calculated on a subpopulation basis (only 17*16/2 = 136 pairs in our example). This is important as the calculation of least-cost paths can be a lengthy process (which mainly depends on the resolution of the cost matrix layer and the number of pairwise paths that needs to be calculated) as it is a computationally expensive process. Note that the analysis is not restricted to the currently implemented genetic differentiation indices. An example how researcher can use their own favorite genetic distance index is provided below (see Section 5: A customised step by step analysis).

2.4 Run the created example

For demonstration purposes we limited the number of subpopulations used in this example to the first six subpopulations. We did this by simply subsetting our genetic data set using the index function "[]". If we add up the number individuals in the first six subpopulations, we find that there are 93 individuals in the first six subpopulations.

```
table(gendata@pop) #individuals per population

##
## P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15
##  10  22  12  23  15  11  14  10   9  11  20  14  13  17  11
## P16 P17
##  12  13

#cumulative sum of the individuals (they are sorted by subpopulations)
cumsum(table(gendata@pop))

## P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15
##  10  32  44  67  82  93 107 117 126 137 157 171 184 201 212
## P16 P17
## 224 237

#so we see there are 93 individuals in the first 4 subpopulations
```

Once you have loaded all your data sets and made sure they are in the right format, the final command to run the LCPMA is very simple, just type:

```
#remember you need to  set mk.pdf=TRUE for a full report
results2 <-  landgenreport(gendata[1:93,], logo, "D",NN=4, mk.pdf=FALSE)

## Compiling report...
## - Landscape genetic analysis using resistance matrices...
## Analysing data ...
## All files are available in the folder:
## C:\Users\s425824\AppData\Local\Temp\RtmpOU2oaT/results
```

3 The results of a least-cost path modelling analysis

This time we want to have a closer look at the `results2` object to understand the output of our analysis. First, we want to create a map using the first six subpopulations and then add the least-cost paths to our analysis. Second, we check if the cost matrix (the R logo) contributes to the explanation of the genetic structure of our subpopulations (checking the results of partial mantel tests and the results of a multiple regression on distance matrices).

To look into the results2 object we need to learn about its slots (components).

```
names(results2)

## [1] "leastcost"
```

This tells us there is a subcomponent leastcost. All of the results of the LCPMA are combined in this subcomponent and they can be accessesed via:

```
names(results2$leastcost)

## [1] "eucl.mat"        "cost.matnames"   "cost.mats"
## [4] "pathlength.mats" "paths"           "gen.mat"
## [7] "mantel.tab"      "mmrr.tab"
```

The reason for the subcomponent "leastcost" is, that the landgenreport function is meant to be extendable. For future implementations of additional analyses, it is more convenient to have the output separated in a subcomponent. The names of the subcomponents are all fairly self-explanatory. For example `eucl.mat` holds the full pairwise matrix of Euclidean distances. You can access it by typing its full name into the console.

```
results2$leastcost$eucl.mat

##       1     2     3     4     5     6
## 1  0.00 24.20 38.10 58.42 23.64 38.74
## 2 24.20  0.00 62.30 76.03 19.19 60.10
## 3 38.10 62.30  0.00 45.05 56.96 23.78
## 4 58.42 76.03 45.05  0.00 82.03 68.08
## 5 23.64 19.19 56.96 82.03  0.00 47.84
## 6 38.74 60.10 23.78 68.08 47.84  0.00
```

As expected this is a matrix with dimensions 6 x 6 (we only used the first 6 subpopulations), which is symmetrical (the distance from subpopulation 1 to subpopulation 6 is the same as from 6 to 1) and has zero entries along the diagonal (the distance from population 1 to population 1 is zero).

In more detail the subsubcomponents are:

- eucl.mat : Euclidean distance matrix
- cost.matnames : names of your cost layers
- cost.mats : the cost matrices (based on your least-cost path algorithm)
- pathlength.mat : the lengths of your least-cost paths (should not be used for an LCPMA, see Etherington & Holland 2013)
- paths : the actual paths as SpatialLines objects
- gen.mat : the genetic distance matrix
- mantel.tab : a table that shows the results of partial Mantel tests ([Wasserman et al., 2010])
- mmrr.tab : a table that shows the results of a multiple matrix regression with randomisation analysis

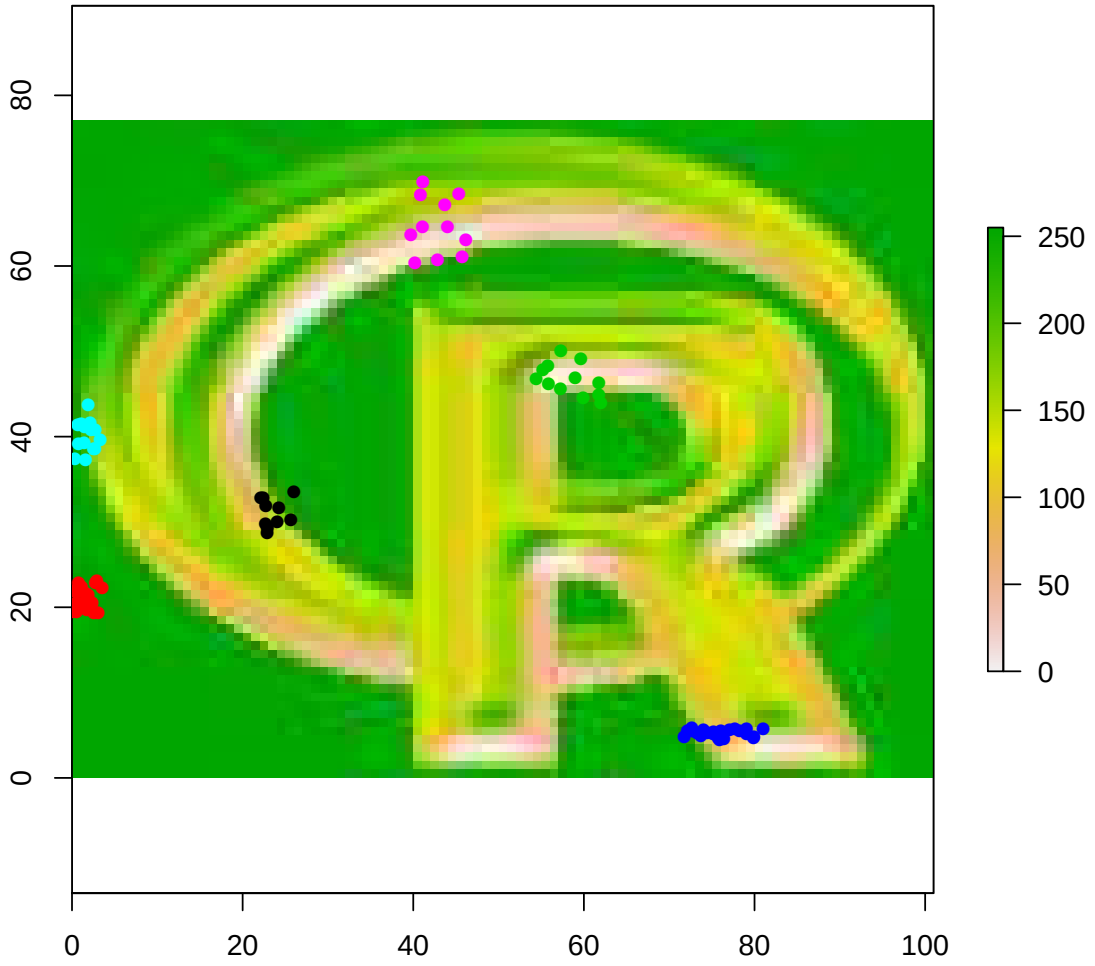
3.1 A map of the least-cost paths

Please note that all of the following plots have already been created by the landgenreport function that you ran earlier and can be found in the indicated results folder. However, to help you better understand the output of the landgenreport function we recreate some of the plots "by hand". This allows you to change the standard output if you think the standard output needs to be customised.

For a simple map of the least-cost path we need three types of information - the least cost layer, the coordinates of the sample individuals for each sub-population and the least-cost

paths. All of this information is within our input data and/or the results2 object. The cost matrix layer is our logo object and we need to find our coordinates for the first six subpopulations. If you remember we put them into our gendata object in the slot @other$xy.

```
plot(logo)
points(gendata@other$xy[1:93,], col=gendata@pop, pch=16)
```



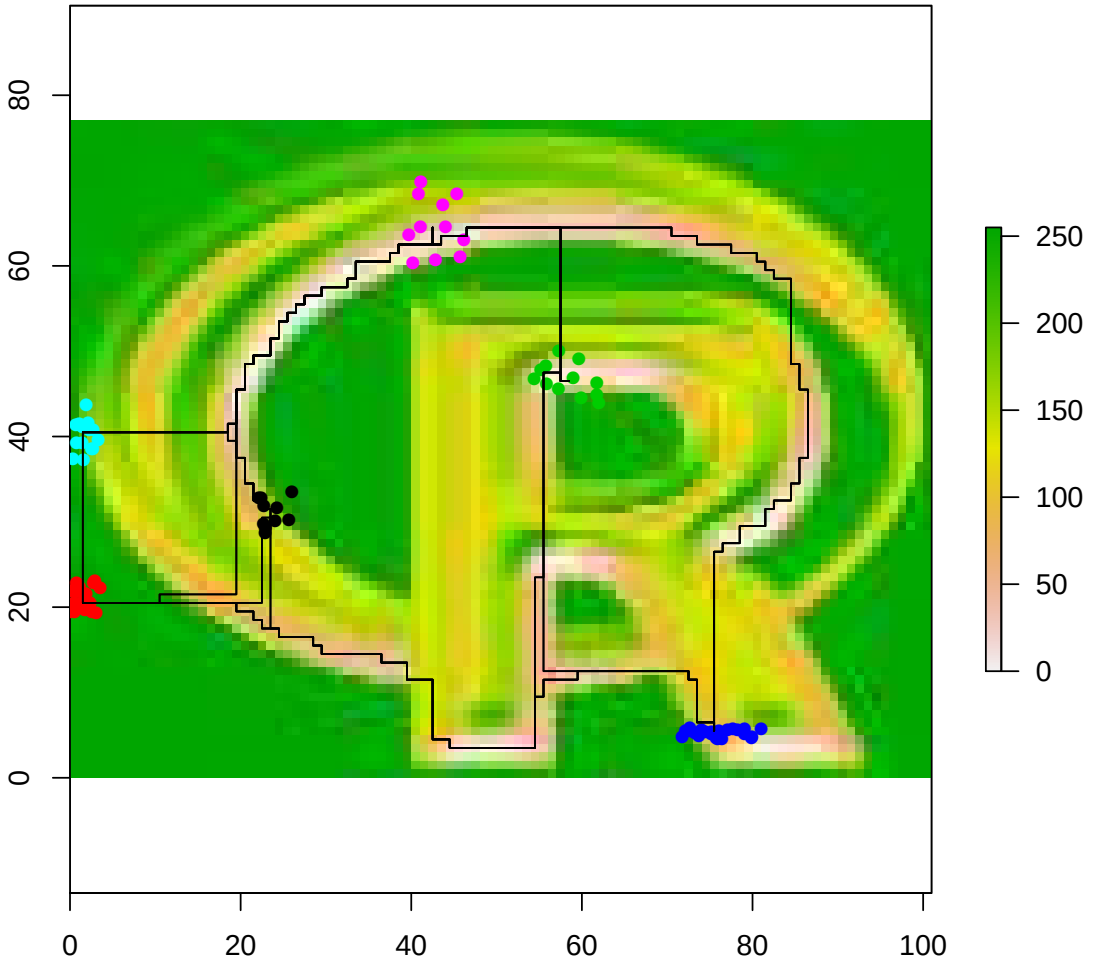
Finally, we need to find the least-cost paths. They are stored in the `paths` subcomponent of our `results2` object. As only one cost matrix layer has been supplied we can access the least-cost path by referencing the first entry in the paths component. We can check how many paths there are by typing:

```
numpaths <- length((results2$leastcost$paths[[1]]))
numpaths

## [1] 15
```

There are 15, which is exactly what we would expect if we'd calculated the number of possible pairs (6*5/2 = 15). To plot the paths we need to do a bit of clever programming. The idea

is simple, we create a loop that prints the paths one at a time. In R this can be done using a for loop or more elegantly using the lapply function (lapply simply applies a function to our path list).

```
plot(logo)
points(gendata@other$xy[1:93,], col=gendata@pop, pch=16)
for (i in 1:numpaths)
{
  lines(results2$leastcost$paths[[1]][[i]])
}

#or more elegant (dummy is just needed to suppress the console output)
dummy <- lapply(results2$leastcost$paths[[1]], function (x) lines(x) )
```



3.2 Checking the results of partial mantel tests and MMRR

To check the results of partial mantel tests and MMRR, we simply access the respective sub-subcomponents by their names:

```
results2$leastcost$mantel.tab

##                   model       r     p
## 1 Gen ~red | Euclidean  0.4424 0.159
## 2 Gen ~Euclidean | red -0.3271  0.81
```

```
results2$leastcost$mmrr.tab

##       layer coefficient tstatistic tpvalue Fstat Fpvalue
## 2       red   2.655e-05      1.709   0.318  1.46   0.512
## 3 Euclidean  -2.165e-03     -1.199   0.424    NA      NA
## 1 Intercept   1.819e-01      2.639   0.719    NA      NA
##       r2
## 2 0.1958
## 3     NA
## 1     NA
```

As expected both analyses indicated that our resistance layer (the R logo) does not explain the population structure of the six subpopulations. Please note that the logo is called "red" in this example, because it is only the red component of the R logo image. The first partial Mantel test tests whether the genetic distance matrix (gen) is correlated with the cost matrix layer (red) controlling for Euclidean distances. This is not the case. Also there is no correlation between the genetic distances (Gen) and the Euclidean distances, controlling for the cost matrix (line 2 of the mantel.tab output). To understand this approach, please refer to Wasserman et al. [2010] and Cushman et al. 2013. The basic idea behind this series of tests is that the partial Mantel test on the correlation between a genetic distance matrix and a cost matrix (controlled by Euclidean distance) should be significant, but the reverse (genetic distance matrix correlated with Euclidean distance matrix controlled for the cost distance matrix) should not be significant.

The MMRR approach tests whether a linear regression between competing distance matrices (Euclidean and cost matrices) is significant, using a randomisation approach to find the test statistics [Wang, 2012]. To access the results of this approach we simply access the subcomponent `mmrr.tab`.

The results are similar to the partial Mantel test. None of the distance matrices can explain the genetic structure between the six subpopulations. The overall correlation (Fstat, Fpvalue and r2 value) is not significant and the distance layers separately do not correlate with the genetic distance matrix (tpvalue for each layer).

Be aware that all of these results can be found in the indicated results folder and also in the complete report if the argument mk.pdf=TRUE was set when running the function landgenreport (check the Appendix for an output of the first example).

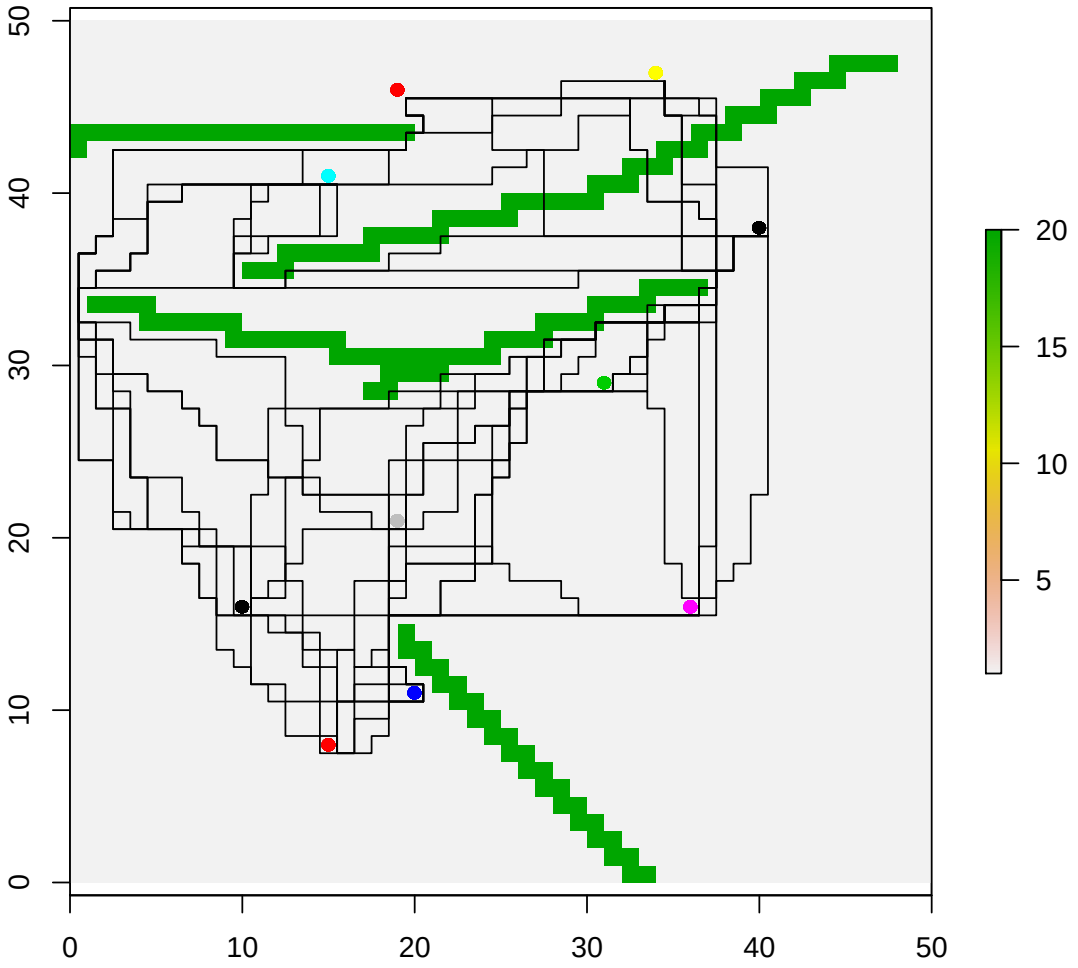
3.3 Output of the first example

To see how the results would look if there was a significant relationship between a cost matrix layer and the pairwise genetic distances, we will now take a look at our first example. The output is stored in the results1 object. Remember we ran:

```
results <-landgenreport(landgen, fric.raster, "D", NN=4, mk.pdf=TRUE)
```

We can rerun the same code as above by simply changing everywhere we typed results2 to results1. This also demonstrates one of the advantages of using a script when running an analysis.

```
#cost matrix layer from example 1
plot(fric.raster)
#what are the coordinates of the samples
points(landgen@other$xy, col=landgen@pop, pch=16)
#plot the least cost paths
dummy <- lapply(results1$leastcost$paths[[1]], function (x) lines(x) )
```




```
#the partial Mantel test results
results1$leastcost$mantel.tab

##                     model       r     p
## 1 Gen ~layer | Euclidean  0.8623 0.002
## 2 Gen ~Euclidean | layer -0.4771 0.993

#the MMRR test results
results1$leastcost$mmrr.tab
```

```
##       layer coefficient tstatistic tpvalue Fstat Fpvalue
## 2     layer    0.012586     11.036   0.001 143.9   0.001
## 3 Euclidean   -0.008834     -3.518   0.011    NA      NA
## 1 Intercept    0.139801      4.505   0.880    NA      NA
##       r2
## 2 0.8727
## 3     NA
## 1     NA
```

3.4 Use the read.genetable function to create a genind object with coordinates in one go

This part of the tutorial demonstrates a different approach to creating a genind object from genetic data stored in a simple CSV table created by a spreadsheet program such as EXCEL or CALC. (also see the examples provided in the PopGenReportIntroduction.pdf)

We use the following data set:

```
platyfile <-  system.file("extdata/platypus1c.csv",package="PopGenReport")
platy <- read.csv(platyfile)
head(platy)

##    ind   pop    lat  long  group age   loci1   loci2
## 1 T158 Black -40.87 145.3 Female juv 141/145 243/243
## 2 T306 Black -40.86 145.3   Male  Ad 145/145 243/243
## 3 T305 Black -40.88 145.3 Female  Ad 143/145 243/243
## 4 T148 Black -40.99 145.4   Male  Ad 141/141 243/243
## 5 T149 Black -40.99 145.4 Female  Ad 141/145 243/245
## 6 T106  Brid -41.23 147.5   Male  Ad 145/147 243/245
##     loci3   loci4   loci5   loci6
## 1 159/173 263/265 215/215 192/192
## 2 159/159 265/267 219/219 192/194
## 3 159/159 265/265 215/215 194/194
## 4 159/159 265/265 215/219 192/194
## 5 159/171 265/267 215/215 194/196
## 6 159/171 267/267 215/215 192/192
```

The first row of the file is a header row. Subsequent rows contain the data for single individuals/samples. The first column contains a unique identifier for the individual/sample. The title for this column must be ”ind”. The second column (optional) is the population that the individual/sample belongs to. The title for the population column (if it is used) must be ”pop”. Separate columns are used for the latitude and longitude coordinates (given in decimal degrees) of each individual/sample (optional) and these columns (if used) should be titled ’lat’ and ’long’. As an optional alternative, Mercator coordinates (or grid points) can be used for spatial coordinates. If this option is used, the column titles should be “x” and “y”. Additional (optional) observations such as gender, age, phenotype, etc. can be placed in a contiguous block of columns. You are free to choose the column titles for these columns, but avoid using spaces and symbols in these titles as they have a tendency to create problems. The remaining columns contain the genetic data. Here you can use a single column per locus (with each of the alleles separated by a defined separator <not commas>) or you can have a single column for each allele. The column titles for the genetic data are up to you, but again we discourage the use of spaces and punctuation marks in the column titles.

It is **very, very** important to follow these formatting instructions closely if you want to import your own data. Keep the spelling and case of the column titles ('ind', 'pop', 'lat', 'long', 'x', and 'y') consistent with the formatting shown in the examples. Additionally, use the same column order as was used in this example. The number of columns can vary (e.g. if you provide more (or less) additional information about your samples or if you have more or less loci). There is some error proofing in the import function, but it is best to stick with the example format as closely as possible. From personal experience, if your dataset isn't being imported correctly, it is most likely because you have made a mistake with the column titles, e.g. something like typing 'latitude', 'Lat', or 'LAT' instead of 'lat'. More detailed instruction and how to access the sample files can be found by typing: `?read.genetable`.

To load the data and to create a genind object from a CSV file that includes individual coordinates, we use the `read.genetable` function from PopGenReport. Here we need to specify which columns contain the required information. As can be seen above the first column has information on the individual identifier, the second has the information on the subpopulation and the third and fourth contain spatial coordinates in latitude and longitude. Columns 5 and 6 have additional information on gender and age and the rest of the columns are our genetic markers (six loci, both alleles are coded in one column, separated by a "/" symbol). You can also specify the character for missing values if it is not "NA" (default value for missing data in R).

```
platy.gen <- read.genetable(platyfile, ind=1, pop=2, lat=3, long=4, other.min=5,
                            other.max=6, oneColPerAll=FALSE, sep="/", ploidy=2)
platy.gen

##
##    ######################
##    ### Genind object ###
##    ######################
## - genotypes of individuals -
##
## S4 class:  genind
## @call: df2genind(X = genes, sep = sep, ncode = ncode, ind.names = inds,
##     loc.names = colnames(genes), pop = pops, missing = missing,
##     ploidy = ploidy)
##
## @tab:  13 x 20 matrix of genotypes
##
## @ind.names: vector of  13 individual names
## @loc.names: vector of  6 locus names
## @loc.nall: number of alleles per locus
## @loc.fac: locus factor for the  20 columns of @tab
## @all.names: list of  6 components yielding allele names for each locus
## @ploidy:  2
## @type:  codom
##
## Optional contents:
## @pop:  factor giving the population of each individual
## @pop.names:  factor giving the population of each individual
##
## @other: a list containing: latlong  data
```

If you want you can explore the content of the data set by accessing its slots (e.g. @pop,

@ind.names). Using `read.genetable`, we created a valid genind object with spatial coordinates in one step. Unfortunately the coordinates are provided as latitude and longitude and they are not useful if you want to calculate pairwise Euclidean distances. Therefore we need to convert them (reproject them) into a coordinate system that is better suited to performing simple Euclidean distance calculations.

4 How to project latlong data into Mercator

Before we convert our spatial coordinates, we use the `mk.map` feature of PopGenReport which creates a map of your samples using a map downloaded from Google maps and your sample spatial coordinates provided as latitudes and longitudes.

```
### remember set mk.pdf=TRUE if you want to have a report !!!!!!
popgenreport(platy.gen, mk.map=TRUE,mk.counts=FALSE, mapdotcolor="red",
             maptype="roadmap", mk.pdf=FALSE)

## Compiling report...
## - Map of individuals...
## Analysing data ...
## All files are available in the folder:
## C:\Users\s425824\AppData\Local\Temp\RtmpOU2oaT/results
## list()
```

To convert the coordinates we need to first specify which coordinate system the coordinates are currently in (for latitudes and longitudes, WGS1984 is the most commonly used format) and which coordinate system they should be converted to (in this case a Mercator projection (see `?Mercator` using the dismo package)). For this exercise we need to load an additional package the dismo package (it should already be loaded from when you loaded the PopGenReport package, but it is better to make sure that it is explicitly loaded into the workspace). We then extract our latlong coordinates from the `@other$latlong` slot (be aware the data needs to be in the order longitude followed by latitude, thus for the current example we reverse the index by setting the columns to 2:1).

```
require(dismo) #load package rgdal
#extract coordinates (long, lat)
longlat<- as.matrix(platy.gen@other$latlong[,2:1])
#projection from latlong to Mercator
xy <-Mercator(longlat)
#add it to platy.gen at @other$xy
platy.gen@other$xy <- xy #again change  and then long
```

For a LCPMA we need to have a cost matrix for the area. For this example, we will simply create one using Google maps (be aware that we are assuming that the colours of the map correctly code for resistance values, which is probably not useful in most cases, but is good enough for demonstration purposes.)

```
e<- extent(range(longlat[,1]), range(longlat[,2])) #extent of the map
tasmap <- gmap(e,style="feature:all|element:labels|visibility:off",
               maptype="roadmap")
names(tasmap) <- "Tasmania"
plot(tasmap)
```

Figure 1: Map on platypus samples in Tasmania

```
points(platy.gen@other$xy, pch=16)
```

Now we can do our LCPMA, this time we want to use Hedrick's Gst as a genetic distance index between subpopulations and the 8 nearest neighbouring cells when calculating the least-cost path. Please refer to the help page of `landgenreport` (`?landgenreport`) for more options.

```
#remember to set mk.pdf=TRUE for a full report
results3 <-landgenreport(platy.gen, tasmap, "Gst.Hedrick", NN=8, mk.pdf=FALSE)

## Compiling report...
## - Landscape genetic analysis using resistance matrices...
## Analysing data ...
## All files are available in the folder:
## C:\Users\s425824\AppData\Local\Temp\RtmpOU2oaT/results

results3$leastcost$mmrr.tab

##       layer coefficient tstatistic tpvalue Fstat Fpvalue r2
```

```
## 1 Intercept   3.195e-01        NaN      NA   NaN      NA  1
## 2  Tasmania  -1.226e-07        NaN      NA    NA      NA NA
## 3 Euclidean   1.971e-05        NaN      NA    NA      NA NA
```

As was done in the prior example, we can plot our least-cost paths to check if they have artefacts (e.g. terrestrial animals dispersing via the sea).

```
#cost matrix layer tasmap
plot(tasmap)
```

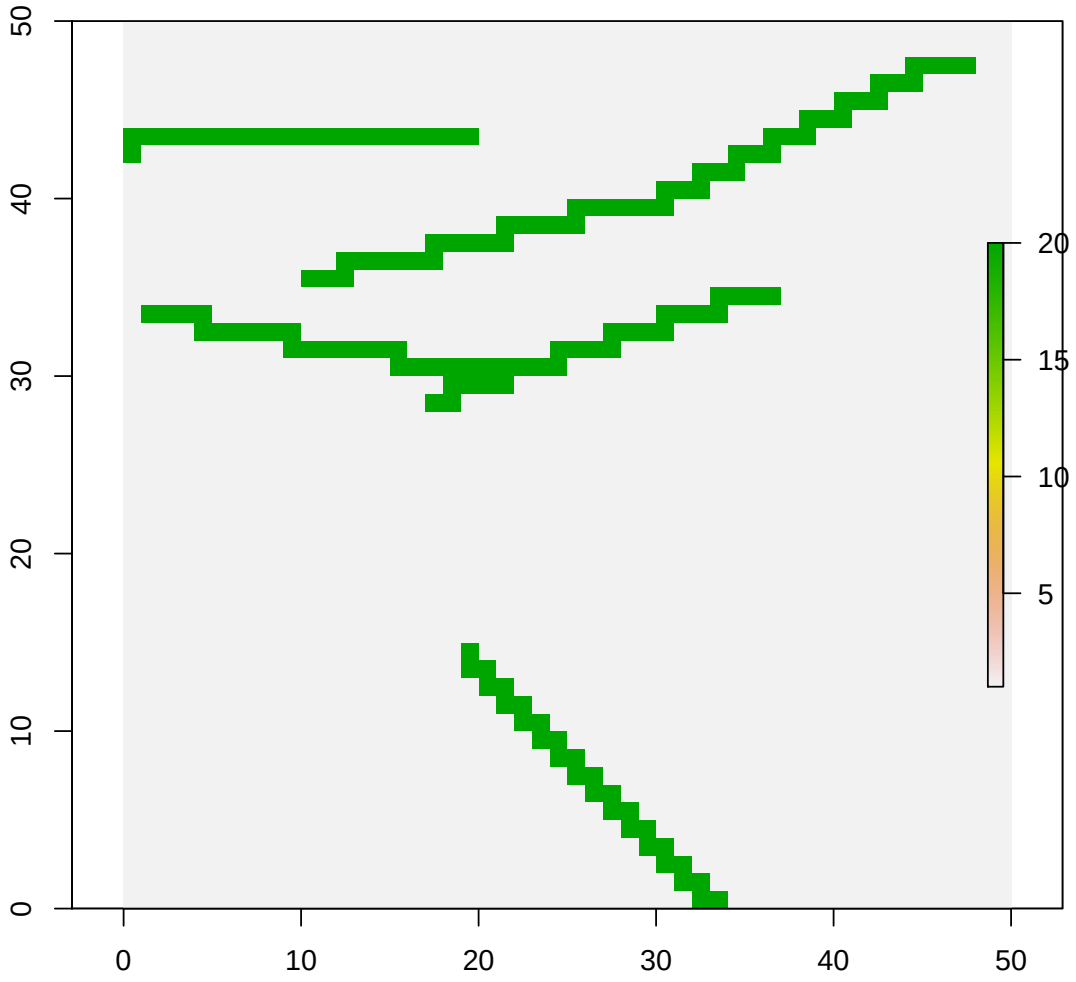
```
#here is where the samples are from
points(platy.gen@other$xy, col=platy.gen@pop, pch=16)
#plot the least cost paths
dummy <- lapply(results3$leastcost$paths[[1]], function (x) lines(x) )
```

5 A customised step by step analysis

This section explains how to do a customised LCPMA analysis. Instead of using the function `landgenreport`, we use the low-level functions `genleastcost` (to calculate a least-cost paths) and `lgrMMRR` (to run a separate multiple matrix regression with randomisation analysis). Additionally, we aim to test a suit of cost matrix layers at the same time. This is often necessary as we do not know the precise resistance values of our cost matrix layers. Thus, we would like to try a number of values and test which resistance values best explain the population structure. For this example, we use the data set that is provided by `landgenreport` (previously used in the first example).

The first step is to create a set of cost matrix layers that we would like to test. To do this we first use `fric.raster` and change the values of the linear structures only. So first we need to explore which values the linear structures initially have.

```
plot(fric.raster)
```

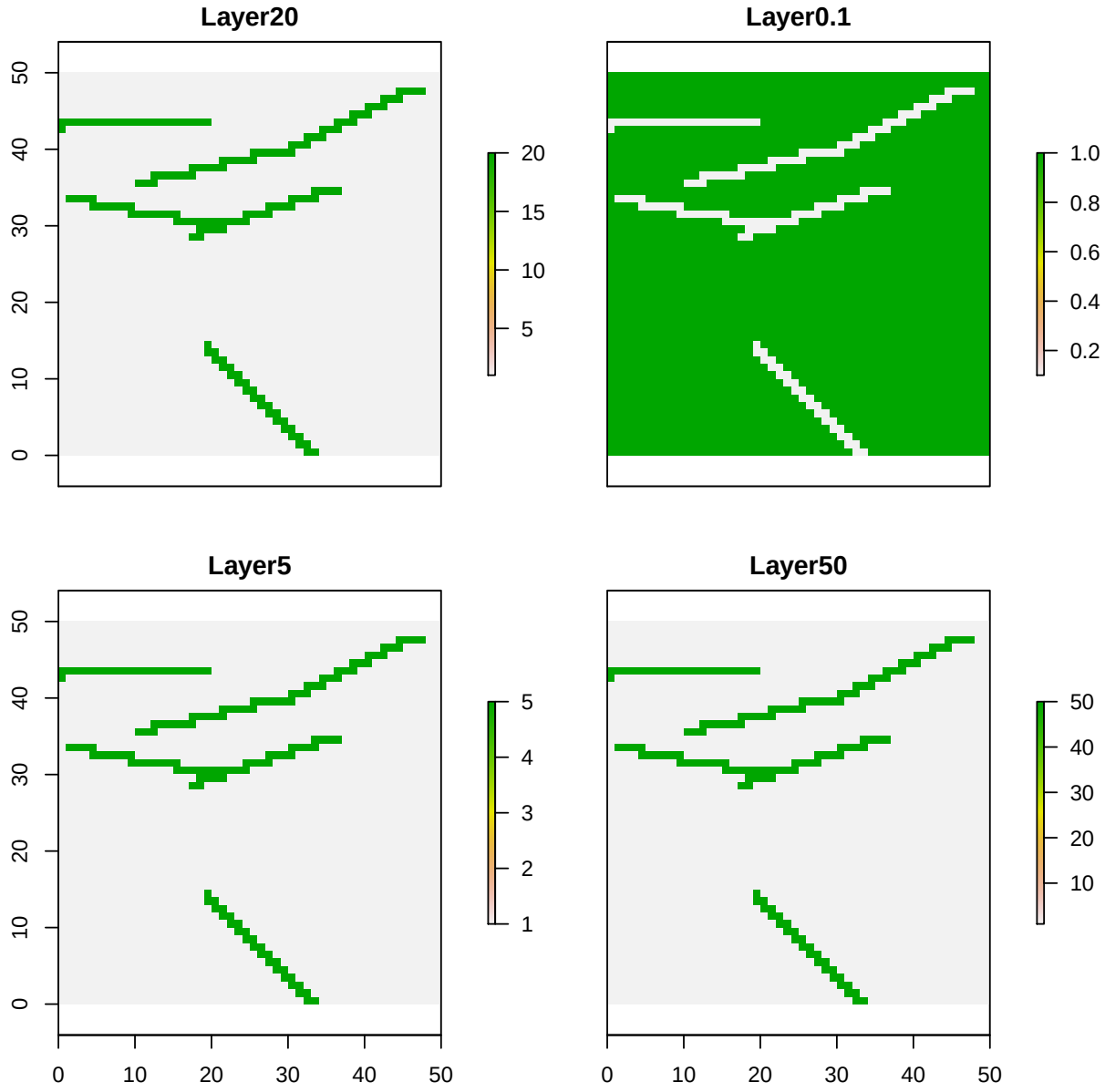



```
table(values(fric.raster))

##
##    1   20
## 2351  149
```

As can be seen from the plot and the matrix output, the matrix cost layer has a value of 1 and the green structures have a value of 20. We will use this information to change the values of the green linear structure to four different values (20 [the original value], 0.1, 5 and 50). Then we stack all these different layers into a single object (called a raster stack) and then run this through our `landgenreport` function.

```
val <-  c(20, 0.1,5,50)

r <- stack(fric.raster)  #the first entry is the original matrix
for (i in 2:4)
{
```

```
r[[i]]  <- fric.raster  #create a copy of fric.raster
r[[i]][values(r[[i]]==20)] <- val[i] #set to new value
names(r[[i]])
}
names(r) <- paste("Layer",val,sep="") #rename layers
plot(r) ###plot all layers (please check the scale)
```



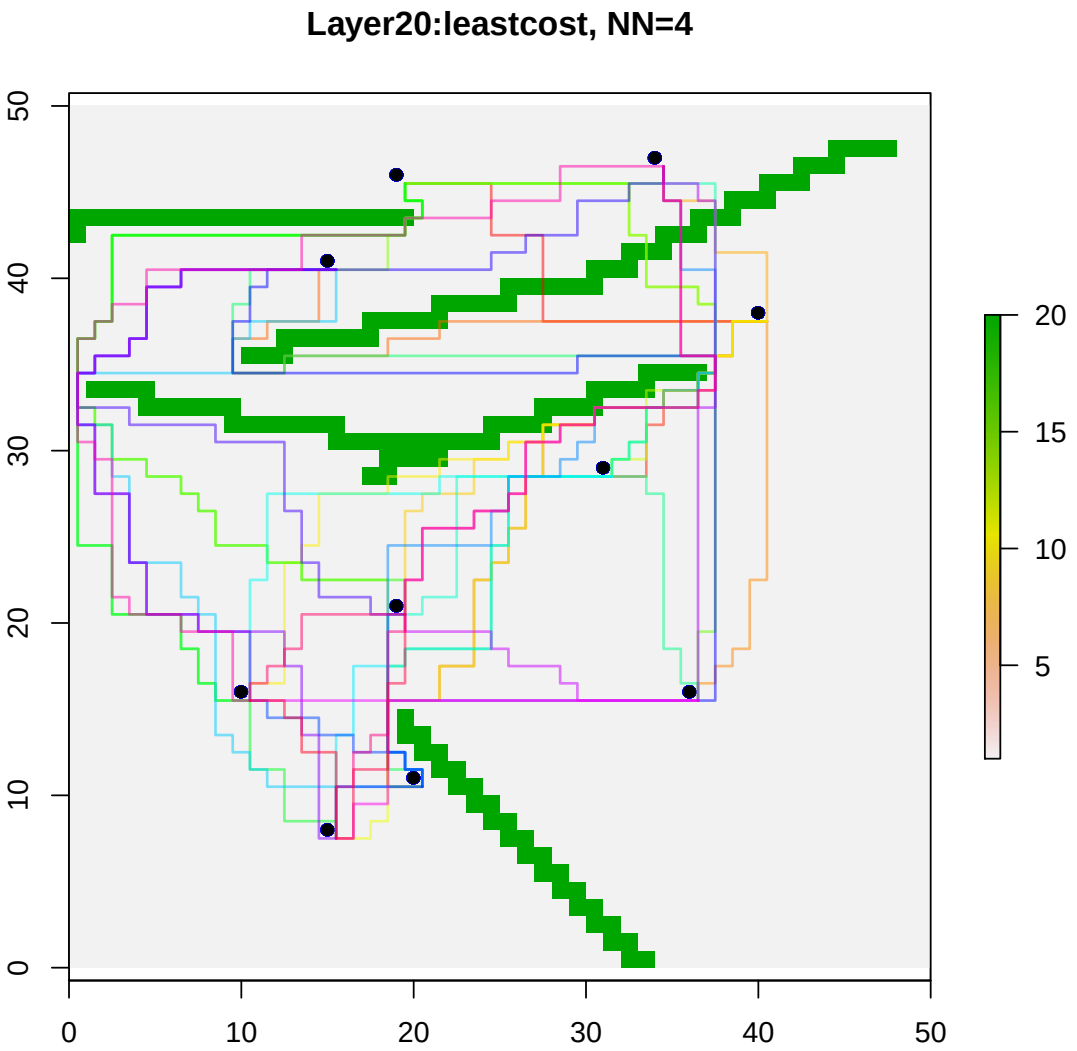
This created the four different layers and combined them into a single raster stack named "r". Please note that values for the green structures are all different as indicated by the scale bar for each plot. Now we can run the LCPMA as before, but using the complete raster stack, using the `landgenreport` function.

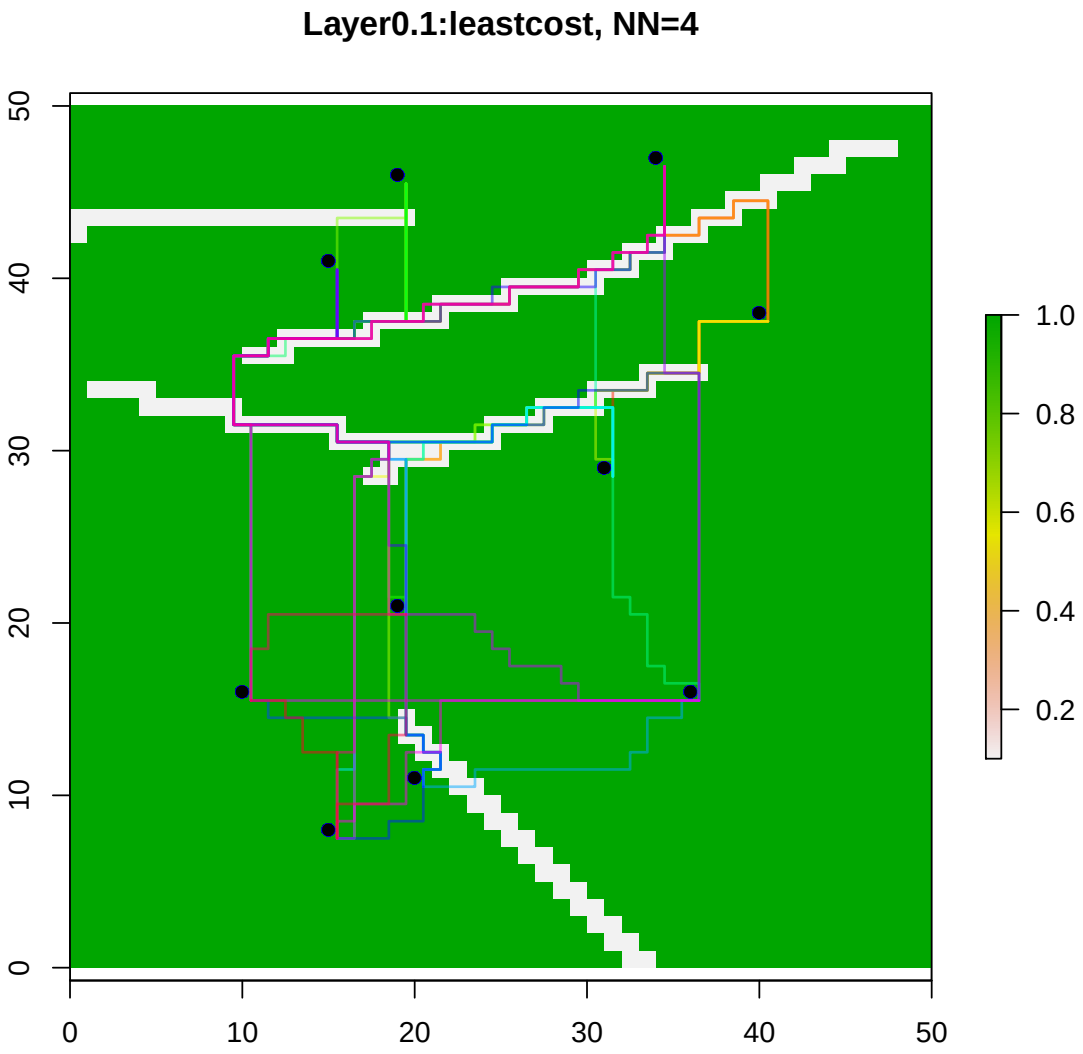
```
results4a <- landgenreport(landgen, r , "Gst.Nei", 4)

## Loci names were not unique and therefore adjusted.
## Compiling report...
## - Landscape genetic analysis using resistance matrices...
## Analysing data ...
```

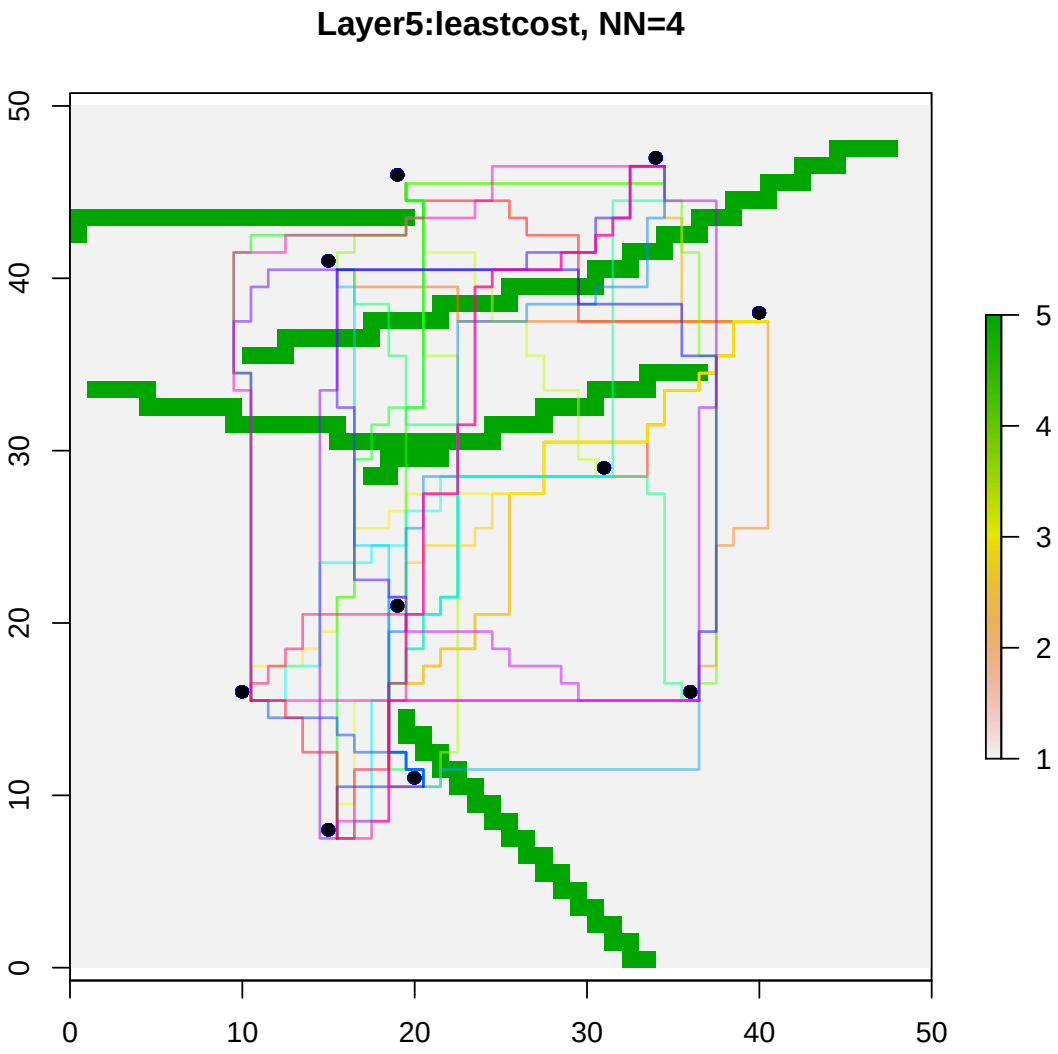
```
## Creating pdf from: LandGenReport.rnw ...

## processing file:  LandGenReport.tex
## output file:  LandGenReport.tex

## Finished.
## Check LandGenReport.pdf for results.
## All files are available in the folder:
## C:\Users\s425824\AppData\Local\Temp\RtmpOU2oaT/results

results4a$leastcost$mmrr.tab

##       layer coefficient tstatistic tpvalue Fstat Fpvalue
## 4    Layer5  -0.0017672    -2.8354   0.015 39.47   0.002
## 6 Euclidean   0.0014612     1.6737   0.118    NA      NA
## 5   Layer50   0.0013100     1.6114   0.184    NA      NA
## 3  Layer0.1  -0.0005682    -1.2720   0.288    NA      NA
## 2   Layer20   0.0003336     0.3725   0.747    NA      NA
## 1 Intercept   0.0151252     3.2739   0.939    NA      NA
##      r2
## 4 0.835
## 6    NA
## 5    NA
## 3    NA
## 2    NA
## 1    NA
```

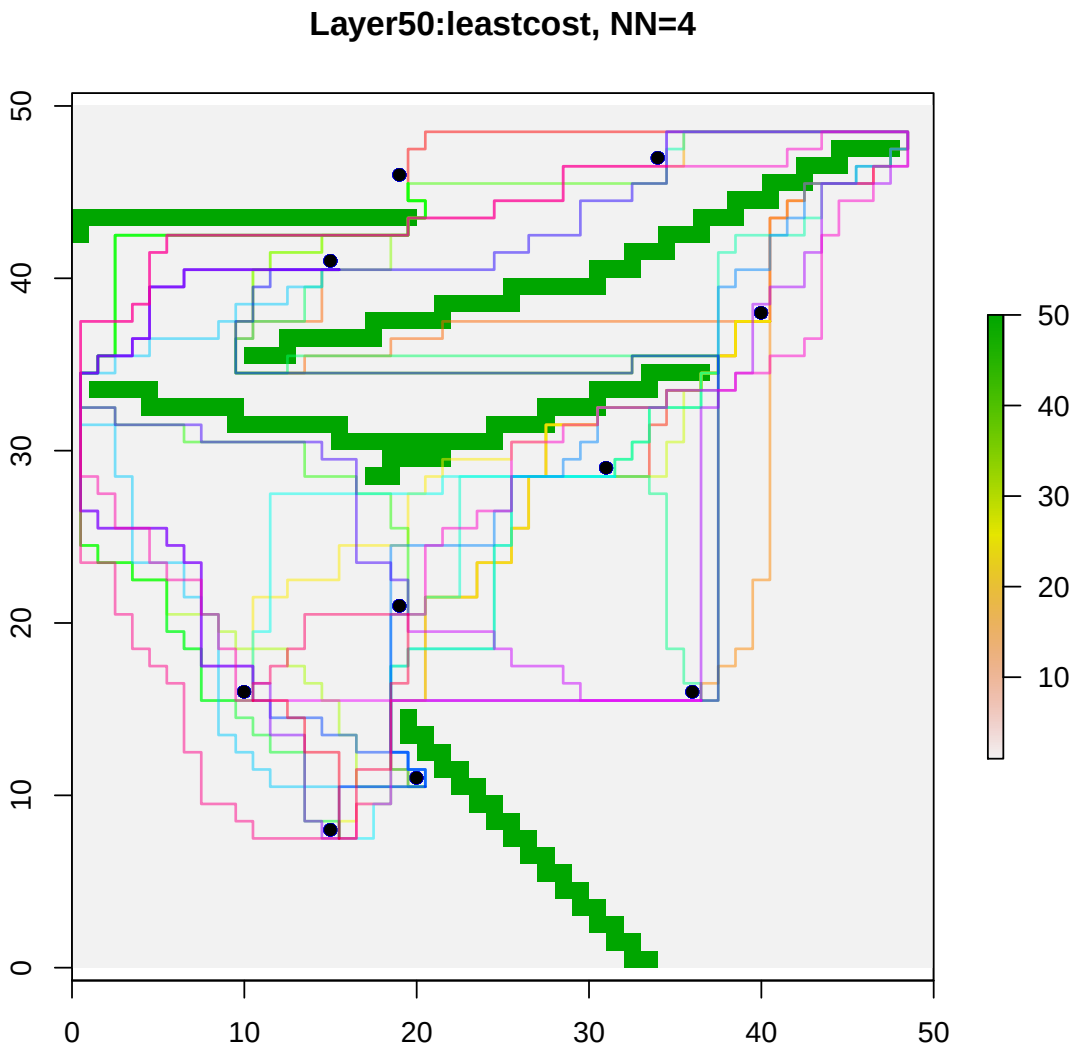
Instead of running the `landgenreport` function we could use the `genleastcost` function. This function offers the same functionality as `landgenreport`, but allows us to run the analysis step by step and makes it possible to customise the analysis (see also `?genleastcost`).

```
results4b <- genleastcost(landgen, r, "Gst.Nei", 4)
```


Layer20:leastcost, NN=4
0
10
20
30
40
50
0
10
20
30
40
50
20
15
10
5


Layer0.1:leastcost, NN=4
0
10
20
30
40
50
1.0
0.8
0.6
0.4
0.2

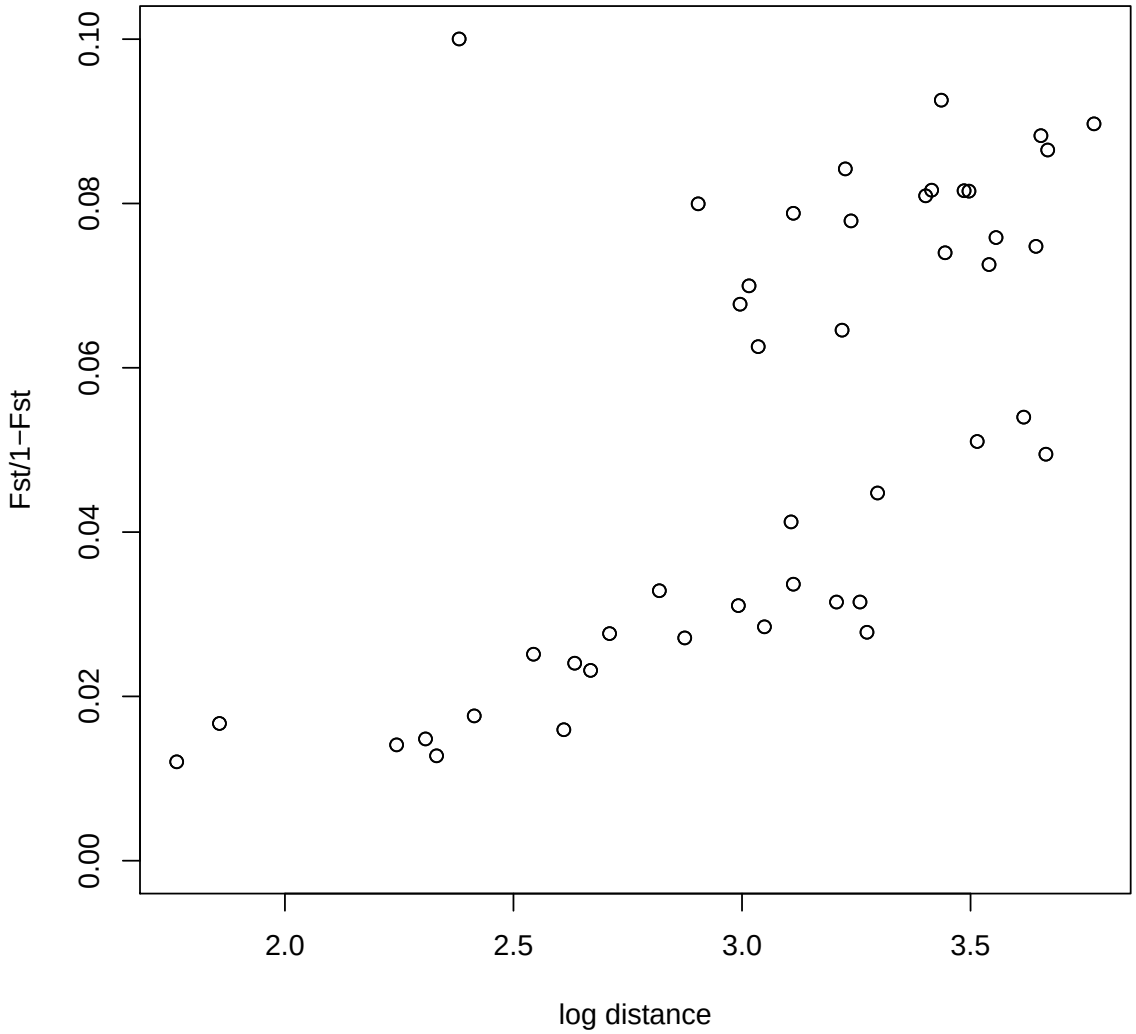

Layer5:leastcost, NN=4
0
10
20
30
40
50
1
2
3
4
5




```
names(results4b)

## [1] "gen.mat"         "eucl.mat"        "cost.matnames"
## [4] "cost.mats"       "pathlength.mats" "paths"
```

A nice feature of genleastcost is that you can observe the progress of the function by looking at the plots that are created as the analysis proceeds. As can be seen, results4b has the same slots as the results produced by the `landgenreport` function (actually without the leastcost level), but the mmrr.tab slot and the mantel.tab slot are missing. Instead of using the 'raw' Gst.Nei index (which is a variant of Fst), we could first visually plot Gst.Nei (actually Gst.Nei / (1-Gst.Nei)) against the log of Euclidean distance, which is the standard transformation used for an isolation by distance plot. For this, we need to extract the values from our results4b object.

```
eucl <- log(results4b$eucl.mat)
gen  <- results4b$gen.mat / (1-results4b$gen.mat)
plot(eucl, gen, ylab="Fst/1-Fst", xlab="log distance")
```



There seems to be a bit of a relationship between both distance matrices. Thus, we use a MMRR that also includes the Euclidean distance matrix. We can use the `lgrMMRR` function, which is based on the MMRR function from Wang [2012]. It requires three arguments, a genetic distance matrix, cost matrix(ces) and a Euclidean distance matrix (see `?lgrMMRR` for details).

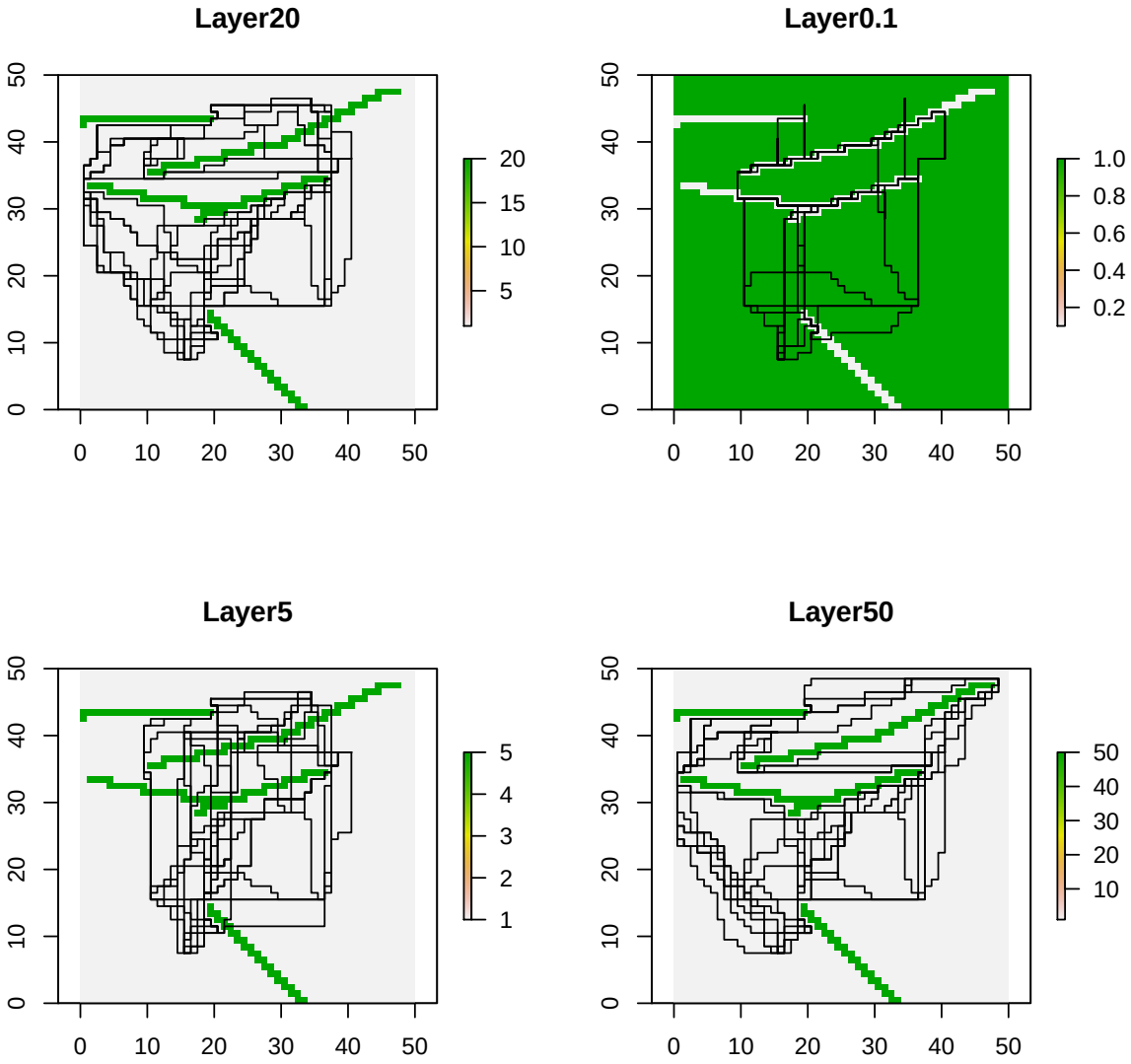
```
lgrMMRR( gen, results4b$cost.mats, eucl)

## $mmrr.tab
##       layer coefficient tstatistic tpvalue Fstat Fpvalue
## 4    Layer5  -0.0014351    -2.9963   0.028 38.28   0.005
## 6 Euclidean   0.0184153     1.6008   0.152    NA      NA
## 5   Layer50   0.0014359     1.5857   0.199    NA      NA
## 3  Layer0.1  -0.0005334    -1.1322   0.358    NA      NA
## 1 Intercept  -0.0195637    -0.8727   0.630    NA      NA
## 2   Layer20   0.0003044     0.3029   0.775    NA      NA
##       r2
## 4 0.8307
## 6     NA
```

```
## 5     NA
## 3     NA
## 1     NA
## 2     NA
```

If you compare this output to the output created by the `landreport` function you can see that they are fairly similar, so the transformation did not affect the result in this case. Layer5 (the green structure has a resistance value of 5) seems to be the cost matrix layer that best describes the data.

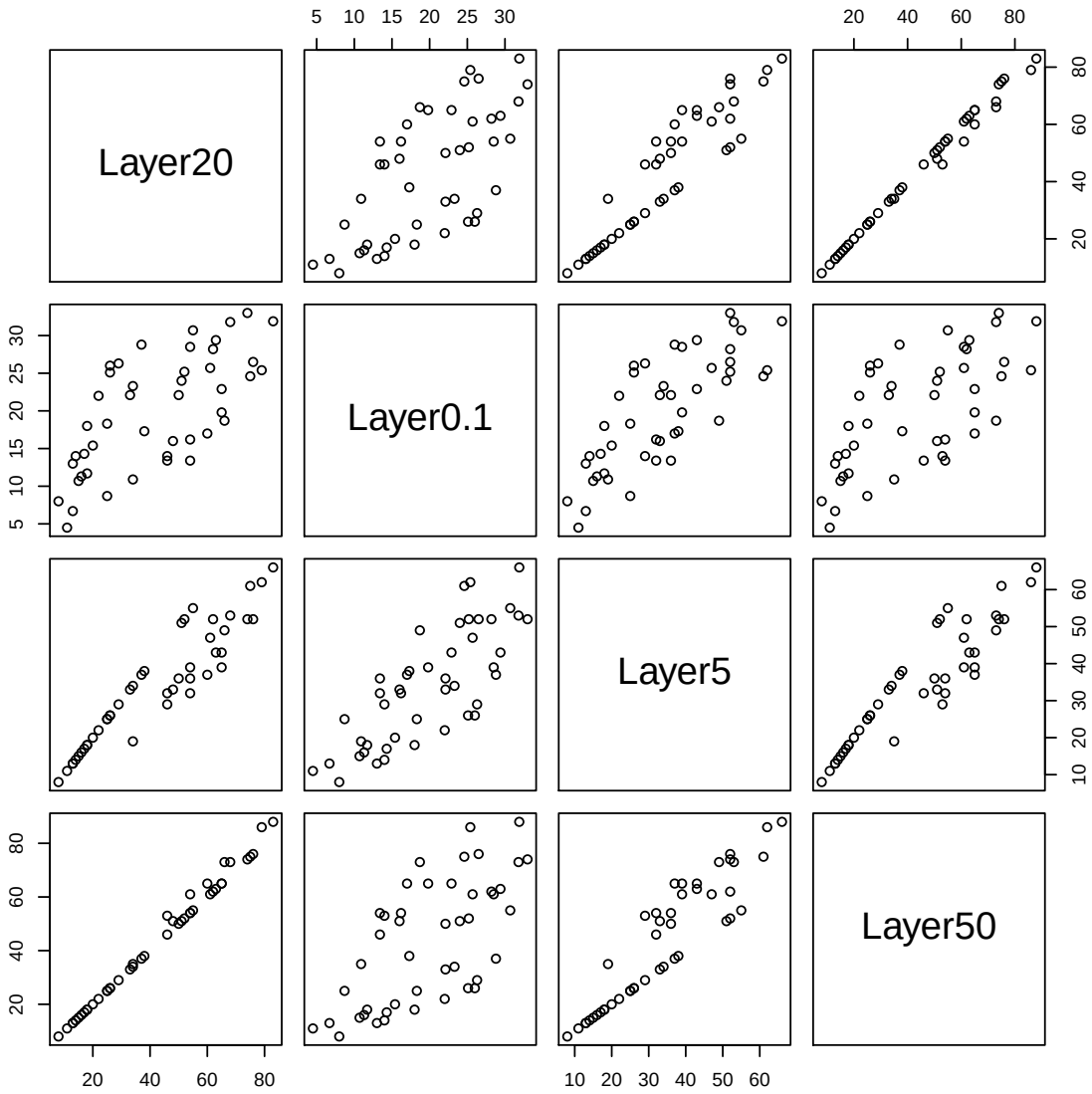
Be aware that all of the cost matrices might be highly correlated to the genetic distance matrix. We can check this by looking at the actual least-cost paths for the four cost matrices. For this we need to slightly extend our script so we can have all of the least-cost paths for each of the cost matrices in one output object.

```
#cost matrix layer from example 1
par(mfrow=c(2,2)) #four plots 2 by 2
for (i in 1:4)
{
plot(r[[i]], main=names(r)[i])
#here are the coordinates stored
points(platy.gen@other$xy, col=platy.gen@pop, pch=16)
#plot the least cost paths
dummy <- lapply(results4a$leastcost$paths[[i]], function (x) lines(x) )
}
```



To calculate the correlation between the cost matrices we need to extract them and put them into the `cor` function. Again this can be achieved elegantly using the lapply function, admittedly a bit confusing because we need to reformat our data, so that it can be processed by the different functions.

```
layers <- data.frame(lapply (results4b$cost.mats,
                             function(x) as.numeric(as.dist(x))))
pairs(layers) #plots the pairwise correlations
```




```
cor(layers)   #calculate r values for each pair

##          Layer20 Layer0.1 Layer5 Layer50
## Layer20   1.0000   0.6749 0.9344  0.9960
## Layer0.1  0.6749   1.0000 0.7848  0.6605
## Layer5    0.9344   0.7848 1.0000  0.9257
## Layer50   0.9960   0.6605 0.9257  1.0000
```

Here we actually see that Layer5, Layer20 and Layer50 are highly correlated $r > 0.9$ and therefore the three matrices should not be used at the same time in the MMRR. We can select only Layer0.1 and Layer20 (the first two from the raster stack) and rerun the `lgrMMRR` function.

```
lgrMMRR( gen, results4b$cost.mats[1:2], eucl)

## $mmrr.tab
##       layer coefficient tstatistic tpvalue Fstat Fpvalue
## 2   Layer20   0.0013048    7.32481   0.009 48.66   0.002
```

```
## 3  Layer0.1  -0.0008133   -1.58141   0.200    NA       NA
## 1 Intercept   0.0167315    0.77580   0.755    NA       NA
## 4 Euclidean  -0.0010827   -0.09928   0.926    NA       NA
##       r2
## 2 0.7807
## 3     NA
## 1     NA
## 4     NA
```

This time only Layer20 comes out as being significant, which is reassuring as in the example a resistance value of 20 was used in a simulation of our populations.

Finally for completeness, we can also run the Wasserman et al. [2010] approach that uses a series of partial Mantel tests. Again we use only the first two layers of the raster stack and the function used is called `wassermann`.

```
wassermann( gen, results4b$cost.mats[1:2], eucl)

## $mantel.tab
##                        model       r     p
## 1   Gen ~Layer20 | Layer0.1  0.8496 0.002
## 3  Gen ~Layer20 | Euclidean  0.7588 0.008
## 6 Gen ~Euclidean | Layer0.1  0.5984 0.001
## 2   Gen ~Layer0.1 | Layer20  -0.341 0.985
## 5 Gen ~Layer0.1 | Euclidean -0.2775 0.948
## 4  Gen ~Euclidean | Layer20 -0.2501 0.942
```

Here Layer20 shows the a significant correlation when corrected for Euclidean distance or Layer0.1, which is not true for the inverse. All other comparison are not following the requirements of Wasserman et al. [2010] (a significant correlation corrected by the competing layer and a non-significant correlation corrected by the original layer), so both methods lead to the same conclusion that Layer20 is the cost matrix that explains best the connectivity in the landscape.

6 Contacts and Citation

Please do not hesitate to contact the authors of `PopGenReport` if you have any comments or questions regarding the package. First of all we are certain there will still be "bugs" in the code as it is impossible to test it under all conditions. Additionally, we would like to expand the capabilities of PopGenReport in the future and comments on features that you would like to have included for your analysis are most welcome. For further updates please check our website: `www.popgenreport.org`.
For citation please use

```
citation("PopGenReport")

##
## Adamack AT and Gruber B (2014). "PopGenReport:
## simplifying basic population genetic analyses in R."
## _Methods in Ecology and Evolution_, *5*(4), pp.
## 384-387.
##
## A BibTeX entry for LaTeX users is
```

```
##
##   @Article{,
##     title = {PopGenReport: simplifying basic population genetic analyses in R},
##     author = {Aaron T. Adamack and Bernd Gruber},
##     year = {2014},
##     journal = {Methods in Ecology and Evolution},
##     volume = {5},
##     number = {4},
##     pages = {384-387},
##     year = {2014},
##   }
```

Have fun running `PopGenReport` and may there be exciting results :-)

Bernd & Aaron
bernd.gruber@canberra.edu.au
aaron.adamack@canberra.edu.au

7 References

References

Ian J Wang. Environmental and topographic variables shape genetic structure and effective population sizes in the endangered Yosemite toad. *Diversity and Distributions*, 18 (10):1033–1041, 2012. doi: 10.1111/j.1472-4642.2012.00897.x. URL `<GotoISI>://WOS:000308466900008`.

Tzeidle N Wasserman, Samuel A Cushman, Michael K Schwartz, and David O Wallin. Spatial scaling and multi-model inference in landscape genetics: Martes americana in northern Idaho. *Landscape Ecology*, 25(10):1601–1612, 2010. doi: 10.1007/s10980-010-9525-7. URL `<GotoISI>://WOS:000283371000011`.

8 Appendix

Output of a full report running: `results <-landgenreport(landgen, fric.raster, "D", mk.pdf=TRUE)`.

A Population Genetic Report

using PopGenReport Ver. 2.0

Adamack & Gruber (2014)

July 10, 2014

Contents

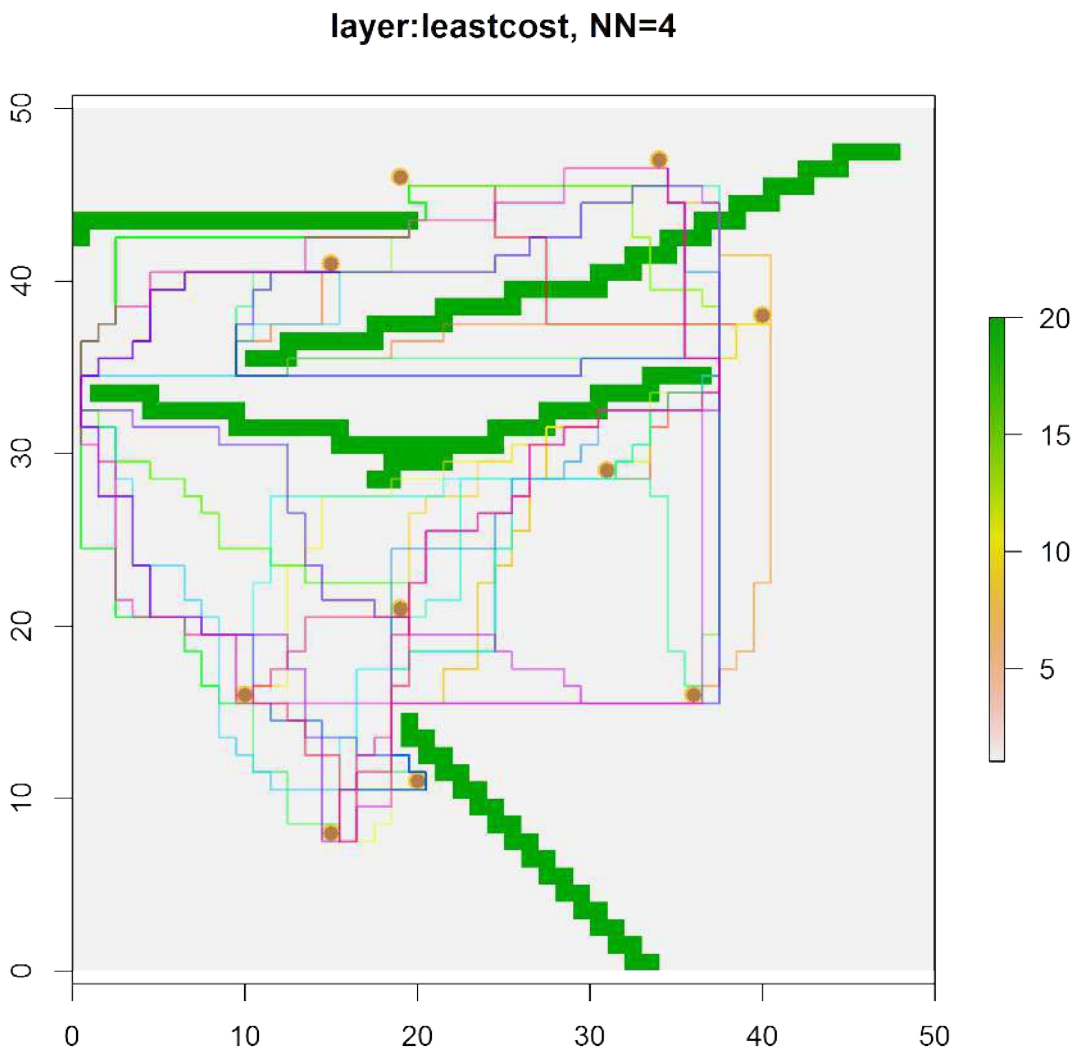
1 Landscape Genetic Analysis **2**
1.1 Maps of resistance matrices 2
1.2 Pairwise Euclidean distances 3
1.3 Pairwise cost distances 5
1.4 Pairwise path lengths 7
1.5 Pairwise genetic distances (D) 9
1.6 Partial Mantel tests following the approach of Wassermann et al. 2010 11
1.7 Multiple Matrix Regression with Randomization analysis 12

1 Landscape Genetic Analysis

Here some initial words on the method....

1.1 Maps of resistance matrices

The following pages show simple maps of the resistance matrices. In case of the pathtype is "leastcost" also the least-cost paths are shown.



1.2 Pairwise Euclidean distances

	1	2	3	4	5	6	7	8	9	10
1	0	22	13	34	25	22	11	27	37	39
2	22	0	21	35	6	34	15	25	31	38
3	13	21	0	21	20	14	18	14	25	26
4	34	35	21	0	30	17	39	10	11	6
5	25	6	20	30	0	33	20	20	25	33
6	22	34	14	17	33	0	31	18	26	22
7	11	15	18	39	20	31	0	30	39	43
8	27	25	14	10	20	18	30	0	10	14
9	37	31	25	11	25	26	39	10	0	9
10	39	38	26	6	33	22	43	14	9	0

Table 1: Pairwise euclidean distances

1.3 Pairwise cost distances

	1	2	3	4	5	6	7	8	9	10
1	0	48	18	51	46	26	34	38	52	55
2	48	0	60	76	11	68	16	65	61	74
3	18	60	0	33	54	18	46	20	34	37
4	51	76	33	0	65	25	79	13	15	8
5	46	11	54	65	0	62	25	54	50	63
6	26	68	18	25	62	0	54	22	26	29
7	34	16	46	79	25	54	0	66	75	83
8	38	65	20	13	54	22	66	0	14	17
9	52	61	34	15	50	26	75	14	0	13
10	55	74	37	8	63	29	83	17	13	0

Table 2: Pairwise cost distances - layer, pathtype='leastcost', NN=4

1.4 Pairwise path lengths

Path lengths are only calculated if path type is "leastcost".

	1	2	3	4	5	6	7	8	9	10
1	0	29	18	51	46	26	15	38	52	55
2	29	0	41	76	11	49	16	65	61	74
3	18	41	0	33	54	18	27	20	34	37
4	51	76	33	0	65	25	60	13	15	8
5	46	11	54	65	0	62	25	54	50	63
6	26	49	18	25	62	0	35	22	26	29
7	15	16	27	60	25	35	0	47	75	64
8	38	65	20	13	54	22	47	0	14	17
9	52	61	34	15	50	26	75	14	0	13
10	55	74	37	8	63	29	64	17	13	0

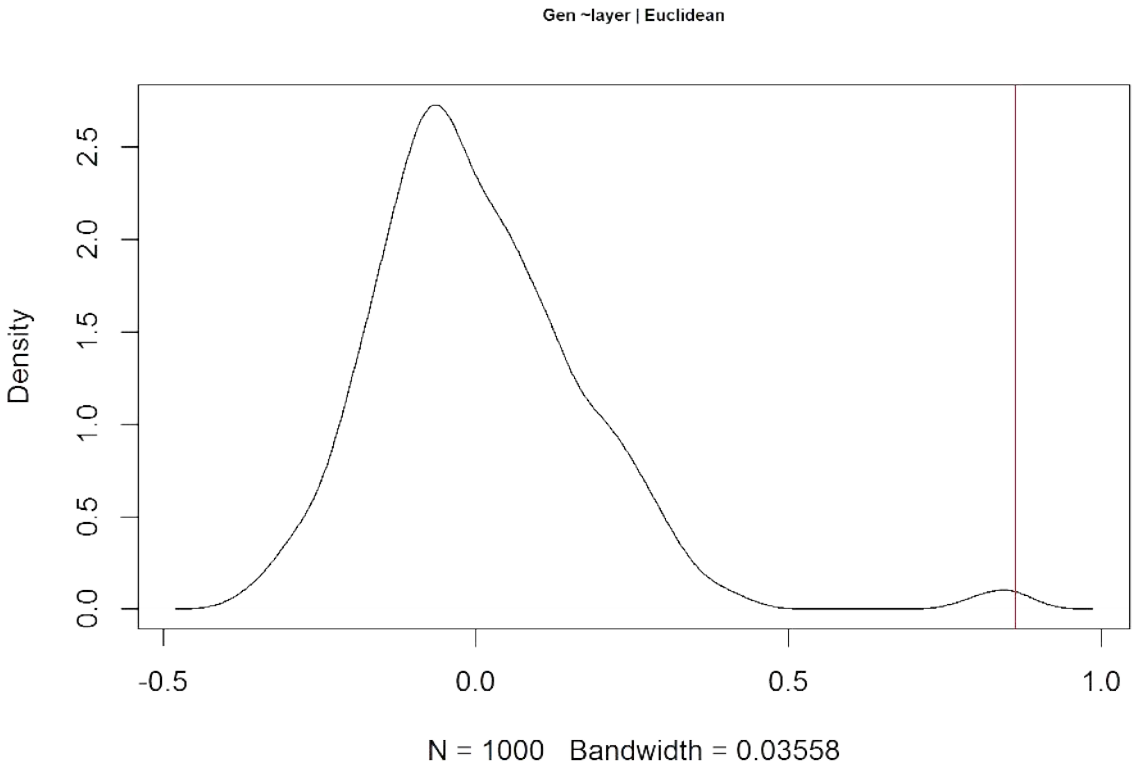
Table 3: Pairwise path lengths (based on least cost paths) - layer, pathtype='leastcost', NN=4

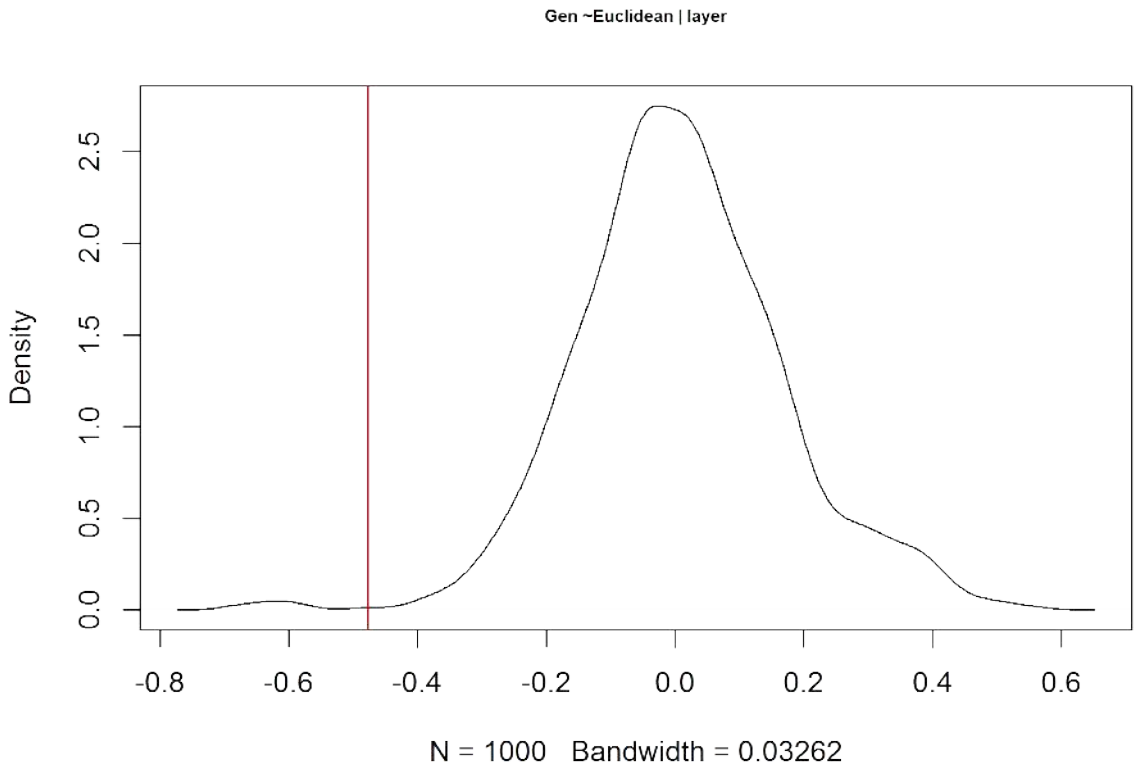
1.5 Pairwise genetic distances (D)

	1	2	3	4	5	6	7	8	9	10
1	0.000	0.640	0.232	0.458	0.645	0.356	0.701	0.435	0.475	0.438
2	0.640	0.000	0.602	0.711	0.133	0.653	0.201	0.657	0.678	0.691
3	0.232	0.602	0.000	0.307	0.612	0.248	0.654	0.273	0.331	0.295
4	0.458	0.711	0.307	0.000	0.719	0.330	0.705	0.170	0.180	0.124
5	0.645	0.133	0.612	0.719	0.000	0.690	0.214	0.667	0.671	0.708
6	0.356	0.653	0.248	0.330	0.690	0.000	0.713	0.296	0.308	0.333
7	0.701	0.201	0.654	0.705	0.214	0.713	0.000	0.696	0.677	0.708
8	0.435	0.657	0.273	0.170	0.667	0.296	0.696	0.000	0.142	0.180
9	0.475	0.678	0.331	0.180	0.671	0.308	0.677	0.142	0.000	0.142
10	0.438	0.691	0.295	0.124	0.708	0.333	0.708	0.180	0.142	0.000

Table 4: Pairwise genetic distance (D)

1.6 Partial Mantel tests following the approach of Wassermann et al. 2010





	model	r	p
1	Gen ~layer | Euclidean	0.8623	0.003
2	Gen ~Euclidean | layer	-0.4771	0.994

Table 5: Mantel tests following methodology of Wassermann et al. 2011

1.7 Multiple Matrix Regression with Randomization analysis

The approach follows the approach of Wang 2013 and Legendre et al. 1994.

	layer	coefficient	tstatistic	tpvalue	Fstat	Fpvalue	r2
2	layer	0.013	11.036	0.001	143.917	0.001	0.873
3	Euclidean	-0.009	-3.518	0.021			
1	Intercept	0.140	4.505	0.877			

Table 6: Multiple Matrix Regression wiht Randomization