R package gdistance: distances and routes on geographical grids (version 1.1-2)

Jacob van Etten

September 28, 2011

1 Introduction

This vignette describes `gdistance`, an R package which provides functionality to calculate various distance measures and routes in heterogeneous geographic spaces represented as grids. Distances are fundamental to geospatial analysis (Tobler 1970). The most commonly used geographic distance measure is the great-circle distance, which represents the shortest line between two points, taking into account the curvature of the earth. However, the great-circle distance does not correspond very well to expected travel time/effort between two points. Travel time and the real distance travelled depend on the means of transport, the mode of route-finding, and the characteristics of landscapes and infrastructure. The great-circle distance could be considered as referring to a special case: goal-directed movement with no obstacles, 'as the crow flies'. Other distance measures are needed when travel is not (or less) goal-directed and landscape characteristics affect movement in a spatially heterogeneous way. Package `gdistance` was created to calculate distances and determine routes using geographical grids (rasters) to represent landscape heterogeneity. It provides the following distance and route calculations.

- The least-cost distance mimics route finding 'as the fox runs', taking into account obstacles and the local 'friction' of the landscape.
- A second type of route-finding is the random walk, which has no predetermined destination ('drunkard's walk'). Resistance distance reflects the travel time from origin to goal of the average (Brownian) random walk (McRae 2006).

- 'Randomised shortest paths' are an intermediate form between shortest paths and Brownian random walks, recently introduced by Saerens et al. (2009).

The functionality of gdistance corresponds to other software like ArcGIS Spatial Analyst, GRASS GIS (r.cost, r.walk functions), and CircuitScape (random walk / resistance distance). The gdistance package also contains specific functionality for geographical genetic analyses. The package implements measures to model dispersal histories first presented by Van Etten and Hijmans (2010). Section 9 below introduces with an example how `gdistance` can be used in geographical genetics.

Package `gdistance` uses functionality from a number of other R packages. The most important among these packages is `raster`. To use `gdistance` and to understand the details of this vignette, the reader has to be familiar with the basic functionality of `raster`.

2 Transition* classes

To make distance calculations as flexible as possible, distances and other measures are calculated in various steps. The central classes in `gdistance` are the S4 classes `TransitionLayer` and `TransitionStack`. Most operations have an object of one of these classes either as input or output.

`Transition*` objects can be constructed from an object of class `RasterLayer`, `RasterStack` or `RasterBrick`. These classes are from `raster`, a memory-efficient and user-friendly R package which contains complete geographical grid functionality. The class `Transition*` takes the necessary geographic references (projection, resolution, extent) from the original `Raster*` object. It also contains a matrix which specifies the probability of movements between cells or, in more general terms, the 'conductance' of inter-cell connections. Each row and column in the matrix represents a cell in the original `Raster*` object. Row/column 1 in the transition matrix corresponds to cell 1 in the original raster, and so on. Cell numbers in rasters go from left to right and from top to bottom. For instance, a 3 x 3 raster would have the following cell numbers:

1 2 3
4 5 6
7 8 9

This raster would produce a 9 x 9 transition matrix with rows/columns numbered from 1 to 9.

Using conductance values may be a bit confusing at first. Other software generally uses friction surfaces to calculate distances. Also, distances

are calculated in terms of accumulated friction or cost. [1] However, the relation between conductance and friction is straightforward: conductance is the *reciprocal* of friction (1/friction). It is not strange to use the word 'conductance' in this context (or to use resistance as a synonym for friction). There is an analogy between random walks on geographical grids and electrical current in a mesh of resistors (McRae et al. 2008). Calculations of 'resistance distance' (see below) take advantage of this analogy. Another advantage of using conductance is that it makes it possible to store the values very efficiently as a so-called *sparse* matrix. Sparse matrices only record the non-zero values and information about their location in the matrix. In most cases, cells are connected only with adjacent cells. Consequently, a conductance matrix contains only a small fraction of non-zero values, which occupy little memory in a sparse matrix format. The package `gdistance` makes use of sparse matrix classes and methods from the package `Matrix`, which gives access to fast procedures implemented in the C language.

A first step in any analysis with gdistance is the construction of an object of the class `Transition*`. The construction of a `Transition*` object from a `Raster*` object is straightforward. We can define an arbitrary function to calculate the conductance values from the values of each pair of cells to be connected. Here, we create a raster with 10 degree cells and set it cells to random values between 0.4 and 0.6. We then create a `TransitionLayer` object. The transition value between each pair of cells is the mean of the two cell values.

```
> library(gdistance)

raster version 1.9-19 (22-September-2011)

> r <- raster(nrows = 18, ncols = 36)
> r <- setValues(r, runif(ncell(r), min = 0.4, max = 0.6))
> r

class       : RasterLayer
dimensions  : 18, 36, 648  (nrow, ncol, ncell)
resolution  : 10, 10  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84
values      : in memory
min value   : 0.4000163
max value   : 0.5995257
```

[1] 'Permeability' is a synonym of conductance. Impedance, resistance, cost and friction are used interchangeably to denote the contrary.

```
> tr1 <- transition(r, transitionFunction = mean,
+     directions = 8)
```

We set the `directions` argument to value 8. This connects all adjacent cells in 8 directions. Cells can also be connected in 4 or 16 connections. In chess terms, setting directions to 4 connects all cells with all possible one-cell rook movements (producing 'Manhattan' distances), while setting directions to 8 connects with one-cell queen movements. With 16 directions, all cells are connected with both one-cell queen movements and one-turn knight movements. This can make distance calculations more accurate.[2]

If we inspect the object we created, we see that the resulting `TransitionLayer` object keeps much information from the original `RasterLayer` object.

```
> tr1

class       : TransitionLayer
dimensions  : 18, 36, 648  (nrow, ncol, ncell)
resolution  : 10, 10  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84
values      : conductance
matrix class: dsCMatrix
```

It is also possible to create asymmetric matrices, in which the conductance from i to j is not always the same as the conductance from j back to i. This is relevant, among other things, for modelling travel in hilly terrain, as shown in Example 1 below. On the same slope, a downslope traveler experiences less resistance than an upslope traveler. In this case, the function to calculate conductance values is non-commutative: $f(i,j) \neq f(j,i)$. The `symm` argument in `transition` needs to be set to `FALSE`.

```
> ncf <- function(x) max(x) - x[1] + x[2]
> tr2 <- transition(r, ncf, 4, symm = FALSE)
> class(transitionMatrix(tr1))

[1] "dsCMatrix"
attr(,"package")
[1] "Matrix"
```

[2]Connecting in 16 directions was inspired by the function r.cost in GRASS 6, and the documentation of this function illustrates nicely why connecting in 16 directions can increase the accuracy of the calculations `http://grass.itc.it/grass64/manuals/html64_user/r.cost.html`. Also, see the section on distance transforms in de Smith et al. (2009).

```
> class(transitionMatrix(tr2))

[1] "dgCMatrix"
attr(,"package")
[1] "Matrix"
```

The sparse matrix class `dsCMatrix` is symmetric and contains only half of the matrix. The class `dgCMatrix` can hold an asymmetric matrix. Different mathematical operations can be done with `Transition*` objects. This makes it possible to flexibly model different components of landscape friction.

```
> tr3 <- tr1 * tr2
> tr3 <- tr1 + tr2
> tr3 <- tr1 * 3
> tr3 <- sqrt(tr1)
```

Operations with more than one object require that the different objects have the same resolution and extent.

Also, it is possible to extract and replace values in the matrix using indices.

```
> tr3[cbind(1:9, 1:9)] <- tr2[cbind(1:9, 1:9)]
> tr3[1:9, 1:9] <- tr2[1:9, 1:9]
> tr3[1:5, 1:5]

5 x 5 sparse Matrix of class "dgCMatrix"

[1,] .        0.5833925 .         .         .
[2,] 0.478976 .         0.6168308 .         .
[3,] .        0.5311842 .         0.4703616 .
[4,] .        .         0.6776534 .         0.5776266
[5,] .        .         .         0.4703616 .
```

Some functions require that `Transition*` objects do not contain any isolated 'clumps'. This can be avoided when creating `Transition*` objects, for instance by giving conductance values between all adjacent cells some minimum value. Also, it can be checked visually. Here are a few ways to visualize a `Transition*` object. You can extract the transition matrix with function `transitionMatrix`. This gives a sparse matrix which can be vizualized with function `image`. This shows the rows and columns of the transition matrix and indicates which has a non-zero value ("connection") as a black dot (Figure 1).

```
> image(transitionMatrix(tr1))
```

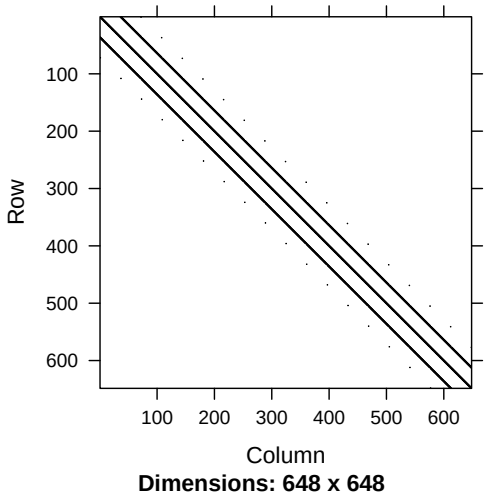

Figure 1: Visualizing a TransitionLayer with function image()

In Figure 1, the central diagonal line represents cell connections between raster cells in the same row. The two adjacent lines represent cells in the same column in the raster, separated by the length of the row in the raster. The curious dots around these lines have to do with the special character of the original raster. Since this raster covers the whole world, the outer meridians touch each other. The software takes this into account and as a result the cells in the extreme left column are connected to the extreme right column! The diagonal connections between the extreme columns explain the isolated dots in Figure 1.

Figure 1 shows which cells contain non-zero values, but gives no further information about levels of conductance. However, we can transform the transition matrix back in to a raster to visualize this. To summarize the information in transition matrix, we can take means or sums across rows or columns, for instance. You can do this with function `raster`. Applied to a `TransitionLayer`, this function converts it to a `RasterLayer`. For the different options see `method?raster("TransitionLayer")`. The default, shown in Figure 2, takes the column-wise means of the non-zero values. All these forms of transformation unavoidably cause information loss, of course.

```
> plot(raster(tr1))
```

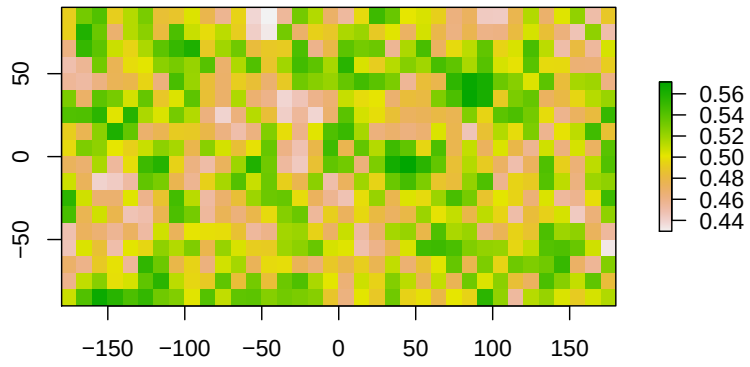

Figure 2: Visualizing a TransitionLayer using the function raster()

3 Correcting inter-cell conductance values

The function `transition` calculates transition values based on the values of adjacent cells in the input raster. However, the centres of diagonally connected cells are more remote from each other than in the case of orthogonally connected cells. Secondly, on equirectangular (lonlat) projection grids, W-E connections are longer at the equator and become shorter towards the poles. Therefore, the values in the matrix need to be corrected for these two types of distortion. Both types of distortion can be corrected by dividing each conductance matrix value between the inter-cell distance. This is what function `geoCorrection` does for us.

```
> tr1CorrC <- geoCorrection(tr1, type = "c", multpl = FALSE)
> tr2CorrC <- geoCorrection(tr2, type = "c", multpl = FALSE)
```

For random walks on longlat grids, there is an additional consideration to be made. The number of connections in N-S direction remains equal when moving from the equator to the poles. This is problematic, because random walks can be seen as analogous to electrical current through a networks of resistors. The inter-cell connections should be thought of as parallel resistors. Moving away from the equator, the inter-meridian space each individual resistor bridges becomes narrower, tending to zero at the poles. Therefore, the

N-S resistance between parallels should decrease when moving away from the equator. The function geoCorrection corrects this distortion by multiplying the N-S transition values with the cosine of the average latitude of the cell centres. This is done with function `geoCorrection`, by setting the argument `type` to "r",

```
> tr1CorrR <- geoCorrection(tr1, type = "r", multpl = FALSE)
```

When similar `Transition*` objects with equal resolution and extent need to be corrected repetitively, computational effort may be reduced by preparing an object that only needs to be multiplied with the `Transition*` object to produce a corrected version of it. The following is equivalent to the previous procedure.

```
> tr1CorrMatrix <- geoCorrection(tr1, type = "r",
+     multpl = TRUE)
> tr1CorrR <- tr1 * tr1CorrMatrix
```

Object `trCorr1Matrix` is only calculated once. It can be multiplied with `Transition*` objects, as long as they have the same extent, resolution, and directions of cell connections. We need to take special care that the geocorrection multiplication matrix (`tr1CorrMatrix`) contains all non-zero values that are present in the `Transition*` object with which it will be multiplied (`tr1`).[3]

4 Calculating distances

With the corrected `Transition*` object we can calculate distances between points. It is important to note that all distance functions require a `Transition*` object with conductance values, even though distances will be expressed in 1/conductance (friction or resistance) units.

To calculate distances, we need to have the coordinates of point locations. This is done by creating a two-row matrix of coordinates. Functions will also accept a `SpatialPoints` object or, if there is only one point, a vector of length two.

```
> sP <- cbind(c(65, 5, -65), c(55, 35, -35))
```

Calculating a distance matrix is straightforward now.

[3] A good alternative is to use geoCorrection(mulpl=FALSE) with a Transition* object with cells connected with value 1.

```
> costDistance(tr1CorrC, sP)

         1        2
2  9805972
3 31745174 22335047

> costDistance(tr2CorrC, sP)

         [,1]     [,2]     [,3]
[1,]        0 11415775 34951321
[2,] 10816497        0 26753977
[3,] 33813271 26327784        0

> resistanceDistance(tr1CorrR, sP)

         1        2
2 3012.742
3 3599.843 3317.169
```

5 Dispersal paths

To determine dispersal paths with a random element, we use the function `passage`. This function can be used for both random walks and randomised shortest paths. The function calculates the number of passages through cells before arriving in the destination cell. Either the total or net number of passages can be calculated. The net number of passages is the number of passages that are not reciprocated by a passage in the opposite direction.

Figure 3 shows the probability of passage through each cell, assuming randomised shortest paths with the parameter theta set to 2. Here we see again how the package takes into account that we are dealing with a raster that covers the whole world. The route crosses the 180 degrees meridian without any problem.

```
> origin <- SpatialPoints(cbind(105, -55))
> goal <- SpatialPoints(cbind(-105, 55))
> rSPraster <- passage(tr1, origin, goal, theta = 2)


> plot(rSPraster)
```

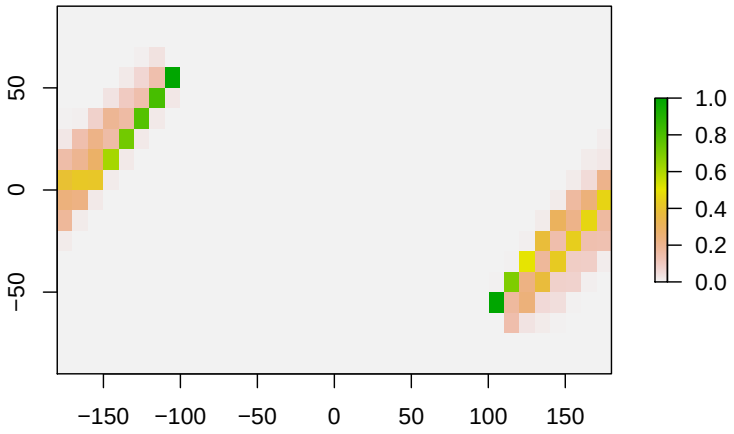

Figure 3: Probability of passage

6 Path overlap and non-overlap

One of the specific uses, for which package gdistance was created, is to look at trajectories coming from the same source (van Etten and Hijmans 2010).

The degree of coincidence of two trajectories can be visualized by multiplying the probabilities of passage.

```
> r1 <- passage(tr1, origin, sP[1, ], theta = 2)
> r2 <- passage(tr1, origin, sP[2, ], theta = 2)
> rJoint <- r1 * r2
```

```
> plot(rJoint)
```

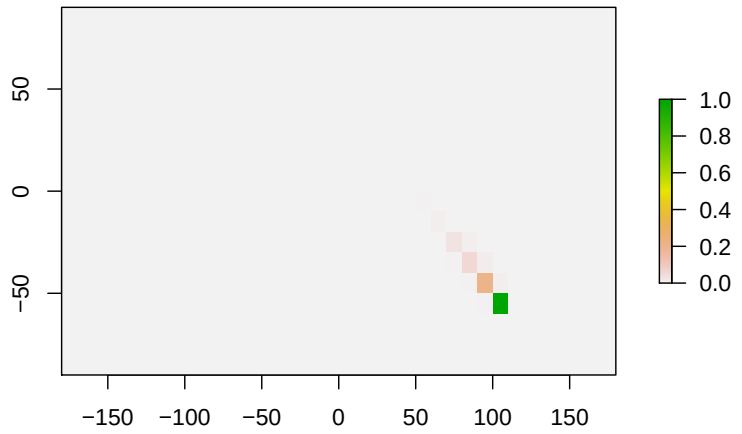

Figure 4: Overlapping part of the two routes

```
> rDiv <- max(max(r1, r2) * (1 - min(r1, r2)) -
+     min(r1, r2), 0)
```

With the function pathInc() we can calculate measures of path overlap and non-overlap for a large number of points. These measures can be used to predict patterns of diversity if these are due to dispersal from a single common source (van Etten and Hijmans 2010). If the argument type contains two elements (divergent and joint), the result is a list of distances matrices.

```
> pathInc(tr1CorrC, sP[1, ], sP[2:3, ], type = c("divergent",
+     "joint"))

$divergent
         1
2 36010415

$joint
         1
2 583857.4
```

```
> plot(rDiv)
```

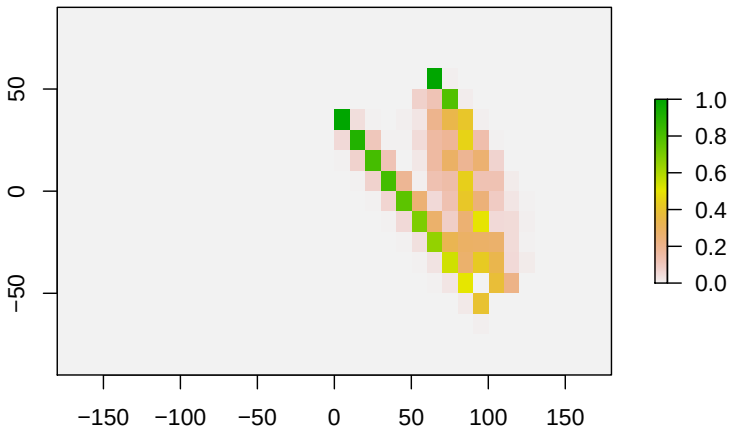

Figure 5: Non-overlapping part of the two routes

7 Example 1: Hiking around Maunga Whau

The previous examples were somewhat theoretical, based on randomly generated values. More realistic examples serve to illustrate the various uses that can be given to this package.

Determining the fastest route between two points in complex terrain is useful for hikers. Tobler's Hiking Function provides a rough estimate for the the maximum hiking speed given the slope of the terrain (Tobler 1993). The maximum speed of off-path hiking (in m/s) is:

speed = exp(-3.5 * abs(slope + 0.05))

Note that the function is not symmetric around 0 (see Figure 6).

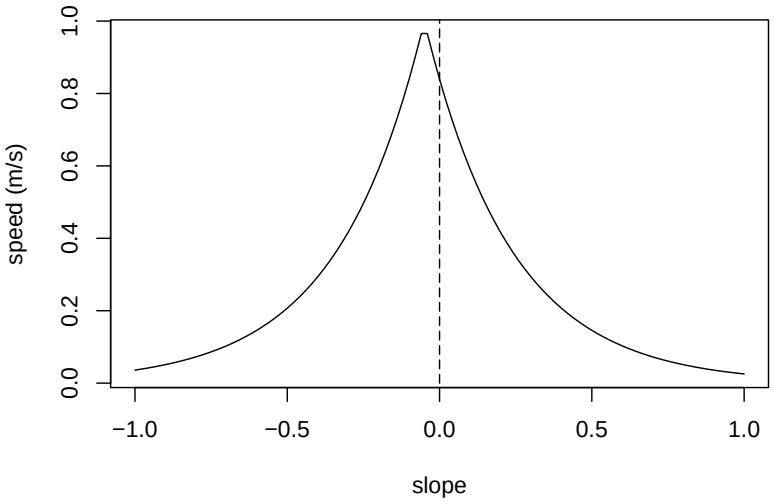

Figure 6: Tobler's Hiking Function

We use the Hiking Function to determine the shortest path to hike around the volcano Maunga Whau (Auckland, New Zealand). First, we read in the altitude data for the volcano. This is a geo-referenced version of the "volcano" data available in Base R datasets (see `?volcano` for more information).

```
> r <- raster(system.file("external/maungawhau.grd",
+     package = "gdistance"))
```

The Hiking Function requires the slope as input.

slope = difference in height / distance travelled

The units of height and distance should be identical. Here, we use meters for both. We calculate the height differences between cells first. Then we use the function `geoCorrection()` to divide by the distance between cells.

```
> heightDiff <- function(x){x[2] - x[1]}
> hd <- transition(r,heightDiff,8,symm=FALSE)
> slope <- geoCorrection(hd, scl=FALSE)
```

Subsequently, we calculate the speed. We need to exercise special care, because the matrix values between non-adjacent cells is 0, but the slope between

these cells is not 0! Therefore, we need to restrict the calculation to adjacent cells. We do this by creating an index for adjacent cells (`adj`) with the function `adjacency()`. Using this index, we extract and replace adjacent cells, without touching the other values.

```
> adj <- adjacency(x = r, fromCells = 1:ncell(r),
+     toCells = 1:ncell(r), directions = 8)
> speed <- slope
> speed[adj] <- exp(-3.5 * abs(slope[adj] + 0.05))
```

Now we have calculated the speed of movement between adjacent cells. We are close to having the final conductance values. Attainable speed is a measure of the ease of crossing from one cell to another on the grid. However, we also need to take into account the distance between cell centres. Travelling with the same speed, a diagonal connection between cells takes longer to cross than a straight connection. Therefore, we use the function `geoCorrection()` again!

```
> x <- geoCorrection(speed, scl = FALSE)
```

This gives our final "conductance" values.

What do these "conductance" values mean? The function `geoCorrection()` divides the values in the matrix with the distance between cell centres. So, with our last command we calculated this:

conductance = speed / distance

This looks a lot like a measure that we are more familiar with:

travel time = distance / speed

In fact, the conductance values we have calculated are the *reciprocal* of travel time.

1 / travel time = speed / distance = conductance

Maximizing the reciprocal of travel time is exactly equivalent to minimizing travel time!

Now we define two coordinates, A and B, and determine the paths between them. We test if the quickest path from A to B is the same as the quickest path from B back to A.

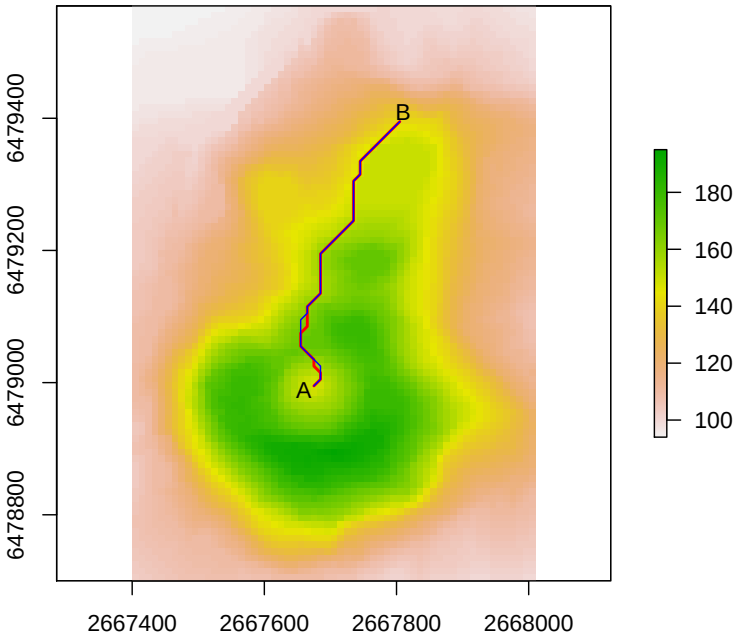

Figure 7: Quickest hiking routes around Maunga Whau

```
> A <- c(2667670, 6479000)
> B <- c(2667800, 6479400)
> AtoB <- shortestPath(x, A, B, output = "SpatialLines")
> BtoA <- shortestPath(x, B, A, output = "SpatialLines")


> plot(r)
> lines(AtoB, col = "red", lwd = 2)
> lines(BtoA, col = "blue")
> text(A[1] - 10, A[2] - 10, "A")
> text(B[1] + 10, B[2] + 10, "B")
```

A small part of the A-B (red) and B-A (blue) lines in the figure do not overlap. This is a consequence of the asymmetry of the Hiking Function.

8 Example 2: Geographical genetics

The direct relation between genetic and geographic distances is known as *isolation by distance* (Wright 1943). Recent work has expanded this relationship to random movement in heterogeneous landscapes (McRae 2006). Also, the geography of dispersal routes can explain observed geospatial patterns of genetic diversity. For instance, diffusion from a single origin (Africa) explains much of the current geographical patterns of human genetic diversity (Ramachandran 2005). As a result, the mutual genetic distance between a pair of humans from different parts from the globe depends on the extent they share their prehistoric migration history.

Within a single continent, however, human genetic diversity may have to do with more recent events. Let's look at diversity in Europe, using the data presented by Balaresque et al. (2010). Within Europe, genetic diversity is often thought to be a result of the migration of early Neolithic farmers from Anatolia (Turkey) to the west.

First we read in the data, including the coordinates of the populations (Figure 8) and mutual genetic distances.

```
> Europe <- raster(system.file("external/Europe.grd",
+     package = "gdistance"))
> Europe[is.na(Europe)] <- 0
> data(genDist)
> data(popCoord)
> pC <- as.matrix(popCoord[c("x", "y")])
```

Then we create three geographical distance matrices. The first corresponds to the great-circle distance between populations. The second is the least-cost distance between locations. Travel is restricted to the land mass. The third is the resistance distance (using the same conductance matrix), which is related to the random-walk travel time between points (McRae 2006).

```
> geoDist <- pointDistance(pC, longlat = TRUE)
> geoDist <- as.dist(geoDist)
> Europe <- aggregate(Europe, 3)
> tr <- transition(Europe, mean, directions = 8)
> tr <- geoCorrection(tr, "c")
> tR <- geoCorrection(tr, "r")
> cosDist <- costDistance(tr, pC)
> resDist <- resistanceDistance(tR, pC)
> cor(genDist, geoDist)
```

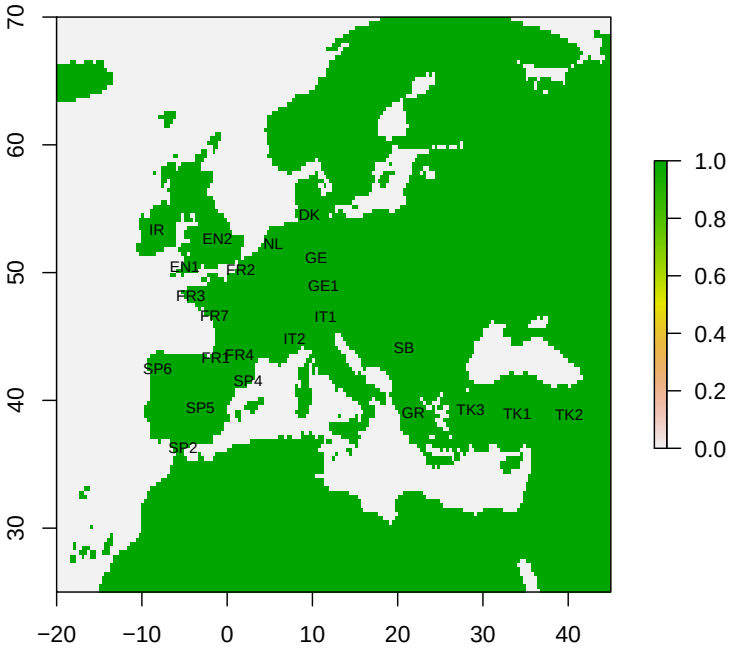

Figure 8: Map of genotyped populations

```
[1] 0.5962655

> cor(genDist, cosDist)

[1] 0.5889319

> cor(genDist, resDist)

[1] -0.05532471
```

Interestingly, the great-circle distance between points turns out to be the best predictor of genetic distance. The other distance measures incorporate more information about the geographic space in which geneflow takes place, but do not improve the prediction. But how well does a wave of expansion from Anatolia explain the spatial pattern?

```
> origin <- unlist(popCoord[22, c("x", "y")])
> pI <- pathInc(tr, origin = origin, fromCoords = pC,
+     type = "joint", theta = 2)
> cor(genDist, pI)

[1] NA
```

At least at first sight, the overlap of dispersal routes explain the spatial pattern better than any of the previous measures. The negative sign of the last correlation coefficient was expected, as more overlap in routes is associated with lower genetic distance. While additional work would be needed to improve predictions and compare the different models more rigorously, the promise of dispersal modelling with `gdistance` is clear.

9 Final remarks

Questions about the use of `gdistance` can be posted on the r-sig-geo email list. Bug reports and requests for additional functionality can be mailed to jacobvanetten@yahoo.com.

10 References

Balaresque P., et al. 2010. A predominantly Neolithic origin for European paternal lineages. *PLoS Biology* 8(1): e1000285.

de Smith, M.J., M.F. Goodchild, and P.A. Longley. 2009. *Geospatial Analysis*. Matador. 3rd edition.

McRae B.H. 2006. Isolation by resistance. *Evolution* 60: 1551–1561.

McRae B.H., B.G. Dickson, and T. Keitt. 2008. Using circuit theory to model connectivity in ecology, evolution, and conservation. *Ecology* 89:2712-2724.

Ramachandran S., et al. 2005. Support from the relationship of genetic and geographic distance in human populations for a serial founder effect originating in Africa. *PNAS* 102: 15942–15947.

Saerens M., L. Yen, F. Fouss, and Y. Achbany. 2009. Randomized shortest-path problems: two related models. *Neural Computation*, 21(8):2363-2404.

Tobler W. 1970. A computer movie simulating urban growth in the Detroit region. *Economic Geography*, 46(2): 234-240.

Tobler W. 1993. Three Presentations on Geographical Analysis and Modeling. `http://www.ncgia.ucsb.edu/Publications/Tech_Reports/93/93-1.PDF`

van Etten, J., and R.J. Hijmans. 2010. A geospatial modelling approach integrating archaeobotany and genetics to trace the origin and dispersal of domesticated plants. *PLoS ONE* 5(8): e12060.

Wright, S. 1943. Isolation by distance. *Genetics* 28: 114–138.