Analysing bathymetric data in `R` with `marmap`

Eric Pante & Benoit Simon Bouhet

September 30, 2015

Contents

---

1 Extracting information from bathymetric data

1.1 Depth and altitude along a transect or path.

Let's start by getting some data into R from the NOAA ETOPO1 database [1]:

```
library(marmap)
papoue <- getNOAA.bathy(lon1 = 140, lon2 = 155,
                        lat1 = -13, lat2 = 0, resolution = 4)
```

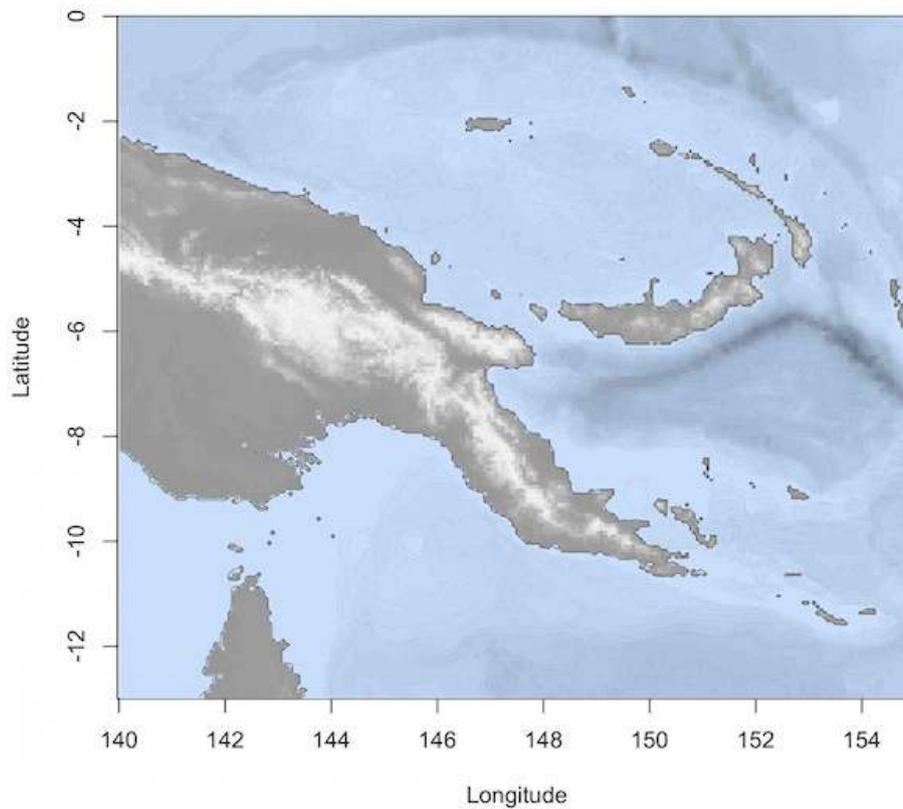We can map these data using `plot.bathy()`:

```
# Creating color palettes
blues <- c("lightsteelblue4", "lightsteelblue3",
           "lightsteelblue2", "lightsteelblue1")
greys <- c(grey(0.6), grey(0.93), grey(0.99))
```

```
plot(papoue, image = TRUE, land = TRUE, lwd = 0.03,
        bpal = list(c(0, max(papoue), greys),
        c(min(papoue), 0, blues)))

# Add coastline
plot(papoue, n = 1, lwd = 0.4, add = TRUE)
```


Basic information about the whole area can be displayed by `summary.bathy()`:

```
summary(papoue)

Bathymetric data of class 'bathy', with 225 rows and 195 columns
Latitudinal range: -12.97 to -0.03 (12.97 S to 0.03 S)
Longitudinal range: 140.03 to 154.97 (140.03 E to 154.97 E)
Cell size: 4 minute(s)

Depth statistics:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -8823   -3092   -1515   -1624      -5    4096

First 5 columns and rows of the bathymetric matrix:
                 -12.9666666666667 -12.9 -12.8333333333333
```

```
140.033333333333                        -36   -36                -36
140.1                                   -37   -36                -36
140.166666666667                        -36   -35                -35
140.233333333333                        -36   -36                -34
140.3                                   -35   -34                -34
                        -12.7666666666667 -12.7
140.033333333333                        -37   -37
140.1                                   -36   -36
140.166666666667                        -35   -36
140.233333333333                        -34   -35
140.3                                   -33   -34
```

We can use the `get.transect()` and `plotProfile()` functions to extract and plot a depth cross section from the `papoue` dataset. `get.transect()` will use the coordinates you input to calculate the coordinates and depths along your transect, and calculate the great circle distance separating each point along the transect from the point of origin (in kilometers).

```
trsect <- get.transect(papoue, 150, -5, 153, -7, distance = TRUE)
head(trsect)

       lon       lat  dist.km depth
1 149.9667 -5.033333  0.00000  -175
2 150.0333 -5.077778  8.88533   100
3 150.1000 -5.122222 17.77024    -9
4 150.1667 -5.166667 26.65472  -223
5 150.2333 -5.211111 35.53877  -367
6 150.3000 -5.255556 44.42238  -270
```
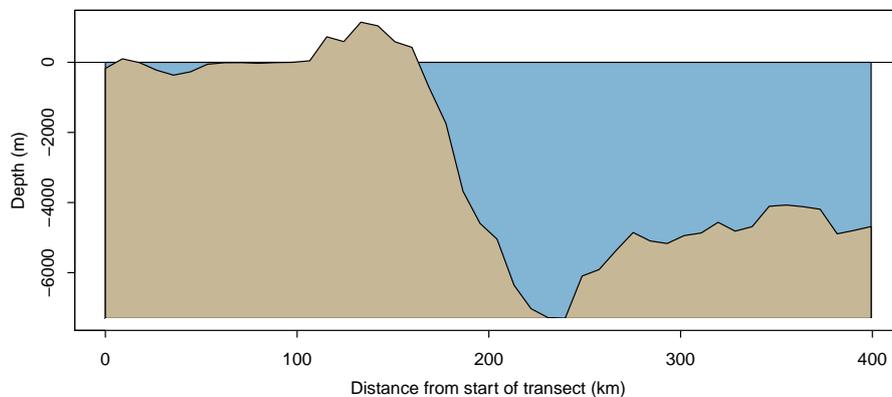
We can plot that information on a map and make a cross section plot with `plotProfile()`. By setting the `locator` option of `get.transect()` to `TRUE`, you can get transect information and make a cross-section plot directly by clicking on a bathemetric map.

```
plotProfile(trsect)
```

The function `path.profile()` takes advantage of both `get.transect()` and `plotProfile()` to retrieve and plot bathymetric information along a path that is not limited to a straight transect between 2 points. See the help file of `plotProfile()` for more details.

1.2 Getting information about points on a bathymetric map

The `get.depth()` function can be used to retrieve depth information by either clicking on the map or by providing a set of longitude/latitude pairs. This is helpfull to get depth information along a GPS track record for instance. If the argument `distance` is set to `TRUE`, the haversine distance (in km) from the first data point on will also be computed. The output will look like this:

```
get.depth(papoue, distance = TRUE)

Waiting for interactive input: click any number of times
on the map, then press 'Esc'

       lon       lat  dist.km  depth
1 146.0200 -2.601702   0.0000   -758
2 147.6167 -1.844152 196.3933   -583
3 149.3193 -2.607345 366.4942  -2121
4 150.7295 -4.249027 553.8867  -2289
```

`get.sample()` can be used in combination with a table containing sampling information to retrieve sample information by clicking on the map. Let's make a fake table of sampling data and use it for plotting and use with `get.sample()`:

```
x <- c(142.1390, 142.9593, 144.0466, 145.9141, 145.9372,
       146.0115, 145.9141, 146.8589, 146.6651, 147.1772,
       147.2856, 152.7475, 152.5025, 152.7816, 152.9010)
y <- c(-2.972065, -3.209449, -3.391399, -4.675720, -4.914153,
       -5.130116, -5.329641, -2.587792, -2.897221, -3.250368,
       -2.720080, -6.005769, -6.211152, -6.326915, -5.990206)

station <- paste("station", 1:15, sep = "")
sampling <- data.frame(x, y, station)
```

We have now created a small table that we can use for further analysis. Let's plot them on a map:

```
head(sampling) # a preview of the first 6 lines of the dataset.

         x         y  station
1 142.1390 -2.972065 station1
2 142.9593 -3.209449 station2
3 144.0466 -3.391399 station3
4 145.9141 -4.675720 station4
5 145.9372 -4.914153 station5
6 146.0115 -5.130116 station6
```
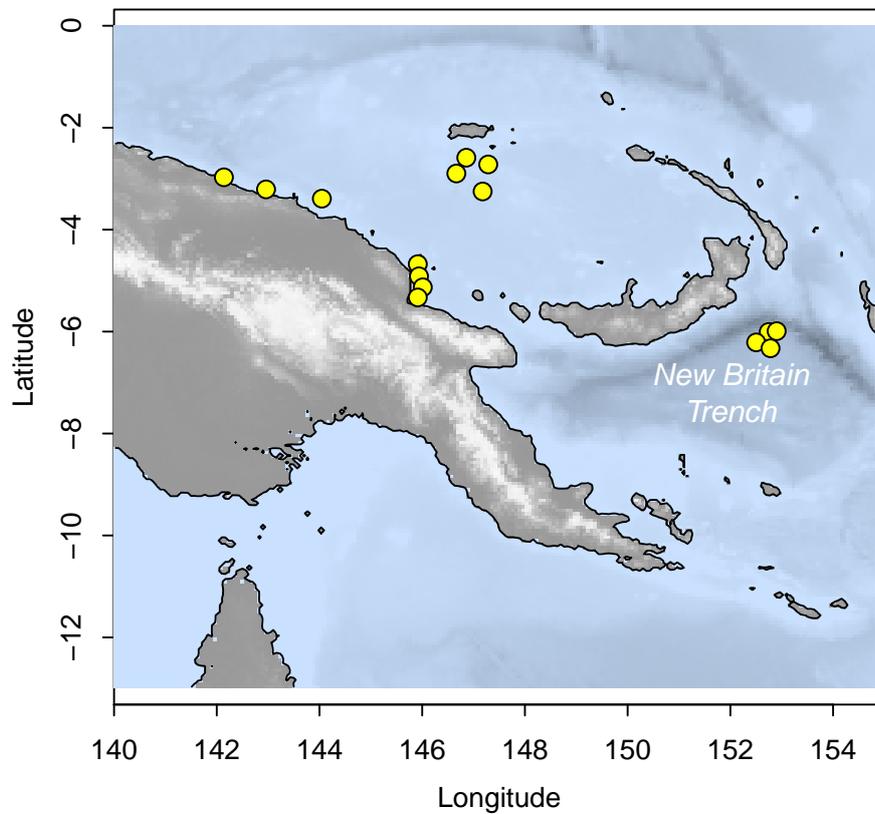
```
plot(papoue, image = TRUE, land = TRUE, n=1,
        bpal = list(c(0, max(papoue), greys),
        c(min(papoue), 0, blues)))

# add sampling points, and add text to the plot:
points(sampling$x, sampling$y, pch = 21, col = "black",
        bg = "yellow", cex = 1.3)
text(152, -7.2, "New Britain\nTrench", col = "white", font = 3)
```


By clicking on the map, we can select the area in the New Britain Trench, to get information on the sampling stations of that area. `get.sample()` will detect that there are samples in the area selected and return the locations of these samples.

```
# click twice on the map to delimit an area:
get.sample(papoue, sampling, col.lon = 1, col.lat = 2)


         x         y  station
12 152.7475 -6.005769 station12
13 152.5025 -6.211152 station13
14 152.7816 -6.326915 station14
```

```
15 152.9010 -5.990206 station15
16 153.2314 -6.023344 station16
```

Instead of using a heat map to represent depth, we can use a simple contour plot for the bathymetry, add a color legend for the depth and associate the color of each point to the desired depth. First, let's get the depth associated with each sampling point in `sampling` using `get.depth()`:

```
# Get the depth for each sampling point
sp <- get.depth(papoue, sampling[,1:2], locator = FALSE)
sp

          lon       lat depth
1   142.1390 -2.972065   -29
2   142.9593 -3.209449  -821
3   144.0466 -3.391399 -1215
4   145.9141 -4.675720   119
5   145.9372 -4.914153 -1265
6   146.0115 -5.130116 -1310
7   145.9141 -5.329641  -955
8   146.8589 -2.587792  -683
9   146.6651 -2.897221 -1422
10 147.1772 -3.250368 -1707
11 147.2856 -2.720080  -653
12 152.7475 -6.005769 -5631
13 152.5025 -6.211152 -4899
14 152.7816 -6.326915 -4272
15 152.9010 -5.990206 -6047
```

Then, create a map, a color legend and add the sampling points:

```
# create a contour plot for the bathymetry and add a scale
par(mai=c(1,1,1,1.5))
plot(papoue, lwd = c(0.3, 1), lty = c(1, 1),
     deep = c(-4500, 0), shallow = c(-50, 0), step = c(500, 0),
     col = c("grey", "black"), drawlabels = c(FALSE, FALSE))
scaleBathy(papoue, deg = 3, x = "bottomleft", inset = 5)

# set color palette for depth
library(shape)
mx <- abs(min(sp$depth, na.rm = TRUE))
col.points <- femmecol(mx)

# plot points and color depth scale
points(sp[,1:2], col = "black", bg = col.points[abs(sp$depth)],
       pch = 21, cex = 1.5)
colorlegend(zlim = c(mx, 0), col = rev(col.points),
            main = "depth (m)", posx = c(0.85, 0.88))
```
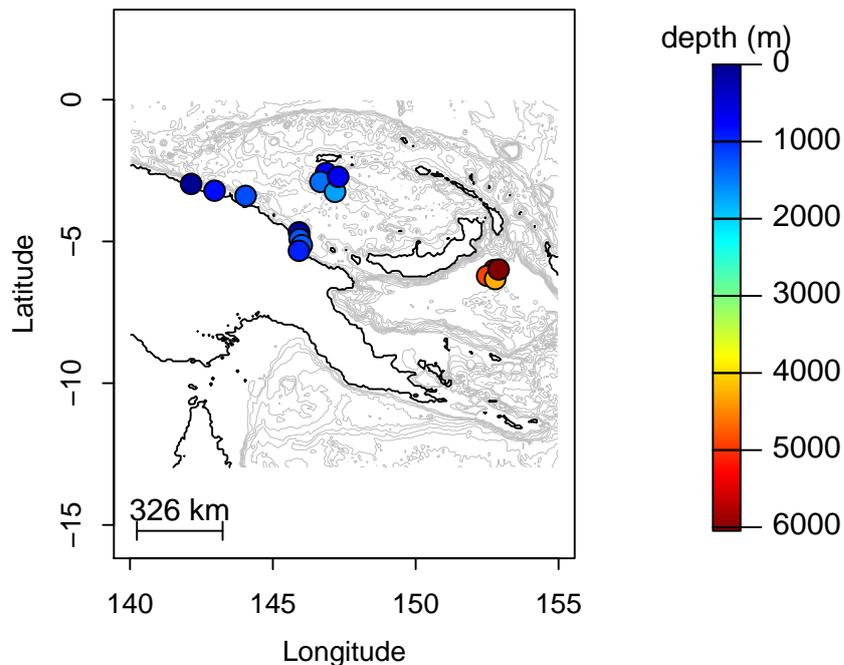
1.3 Computation of projected surfaces

The function `get.area()` can be used to calculate projected surface areas (the projecting surface being the ocean surface). This functions depends on the `geosphere` package [9]. For example, in the case of the Hawaiian Archipelago, we can calculate the surface area of the bathyal (1,000 to 4,000 m) and abyssal regions (4,000 to about 6,000 m).

```
data(hawaii)
bathyal <- get.area(hawaii, level.inf = -4000, level.sup = -1000)
abyssal <- get.area(hawaii, level.inf = min(hawaii),
                    level.sup = -4000)
ba <- round(bathyal$Square.Km, 0)
ab <- round(abyssal$Square.Km, 0)
```

The function `get.area()` returns a list of 4 elements. The surface area in square kilometers (`$Square.Km`), a matrix of zeros and ones delimiting the area of interest (`Area`) and 2 vectors (`$Lon` and `$Lat`) containing the longitudes and latitudes of the area of interest. Such lists can be used to highlight the projected surfaces on an existing bathymetric map using function the `plotArea()`:

```
plot(hawaii, lwd = 0.2)
col.bath <- rgb(0.7, 0, 0, 0.3)
col.abys <- rgb(0.7, 0.7, 0.3, 0.3)

plotArea(bathyal, col = col.bath)
plotArea(abyssal, col = col.abys)
```
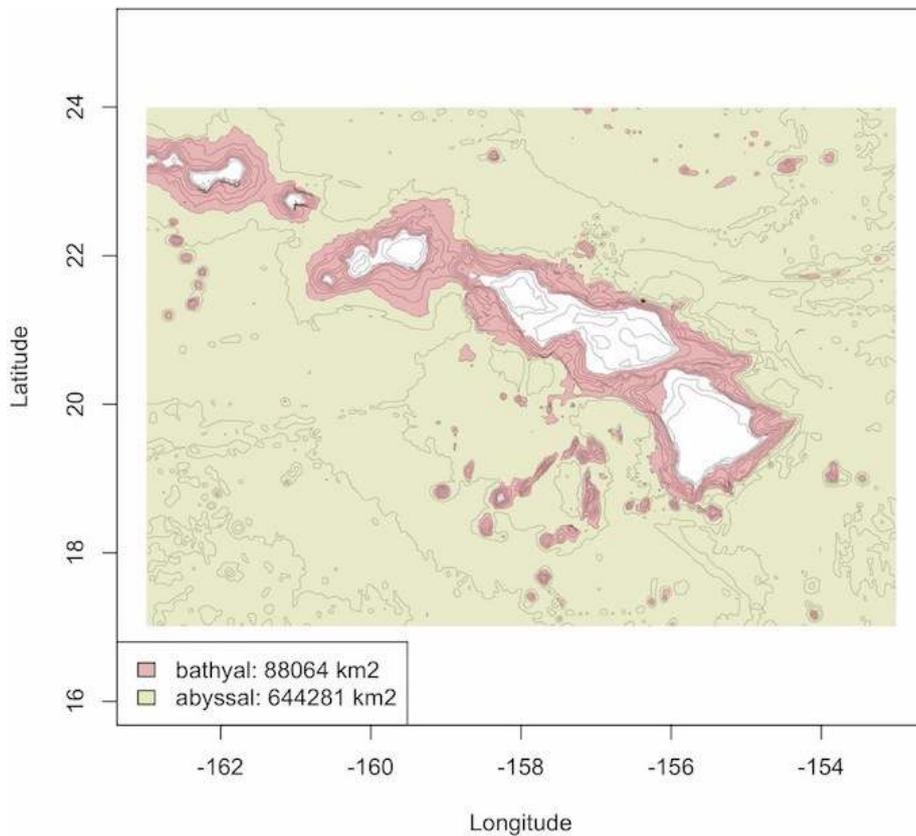
Finally, we can add a legend with the calculated surface for both areas:

```
legend(x="bottomleft",
       legend=c(paste("bathyal:",ba,"km2"),
                paste("abyssal:",ab,"km2")),
       col="black", pch=21,
       pt.bg=c(col.meso,col.bath,col.abys))
```


2 Computing distances

2.1 Using bathymetric data for least-cost path analysis

marmap contains functions to facilitate least-cost path analysis that are based
on the raster [10] and gdistance [8] packages. gdistance calculates routes

8

in a heterogeneous landscape, taking obstacles into account. These obstacles can be defined in marmap based on bathymetric data. We will use the Hawaiian islands as our playground for this section.

```
data(hawaii, hawaii.sites)
sites <- hawaii.sites[-c(1,4),]
rownames(sites) <- 1:4
```

We first compute a transition matrix to be used by `lc.dist()` to compute least cost distances between locations. The transition object generated by `trans.mat()` contains the probability of transition from one cell of a bathymetric grid to adjacent cells, and depends on user defined parameters. `trans.mat()` is especially usefull when least cost distances need to be calculated between several locations at sea. The default values for arguments `min.depth` and `max.depth` of `trans.mat()` ensure that the path computed by `lc.dist()` will be the shortest path possible at sea avoiding land masses. The path can be constrained to a given depth range by setting manually `min.depth` and `max.depth`. For instance, it is possible to limit the possible paths to the continental shelf by setting `max.depth=-200`. Inaccuracies of the bathymetric data can occasionally result in paths crossing land masses. Setting `min.depth` to low negative values (e.g. -10 meters) can limit this problem.

Here, `trans1` is a transition object constrained only by land masses. `trans2` is a transition object that makes travel impossible in waters shallower than 200 meters depth. This step takes a little time.

```
trans1 <- trans.mat(hawaii)
trans2 <- trans.mat(hawaii, min.depth = -200)
```

We can now use these transition objects to calculate least cost distances for `trans1` and `trans2`. The output of `lc.dist()` is a list of geographic positions corresponding to the least-cost path.

```
out1 <- lc.dist(trans1, sites, res = "path")

|==================================================| 100%

out2 <- lc.dist(trans2, sites, res = "path")

|==================================================| 100%
```

We use the `lapply()` function to extract information from these lists and plot lines. Thick orange lines correspond to least-cost paths only constrained by landmasses. Thin black lines are paths constrained by the 200 m isobath. We store the result of `lapply()` in a `dummy` variable to avoid printing of unnecessary information. The coastline is in black, the 200 m isobath is in blue, and isobaths between 5000 and 200 m depth are in grey. Our sampling points are in blue.

```
plot(hawaii, xlim = c(-161, -154), ylim = c(18, 23),
     deep = c(-5000, -200, 0), shallow = c(-200, 0, 0),
     col = c("grey", "blue", "black"), step = c(1000, 200, 1),
```

```
    lty = c(1, 1, 1), lwd = c(0.6, 0.6, 1.2),
    draw = c(FALSE, FALSE, FALSE))
points(sites, pch = 21, col = "blue", bg = col2alpha("blue", .9),
    cex = 1.2)
text(sites[,1], sites[,2], lab = rownames(sites),
    pos = c(3, 4, 1, 2), col = "blue")
lapply(out1, lines, col = "orange", lwd = 5, lty = 1) -> dummy
lapply(out2, lines, col = "black", lwd = 1, lty = 1) -> dummy
```
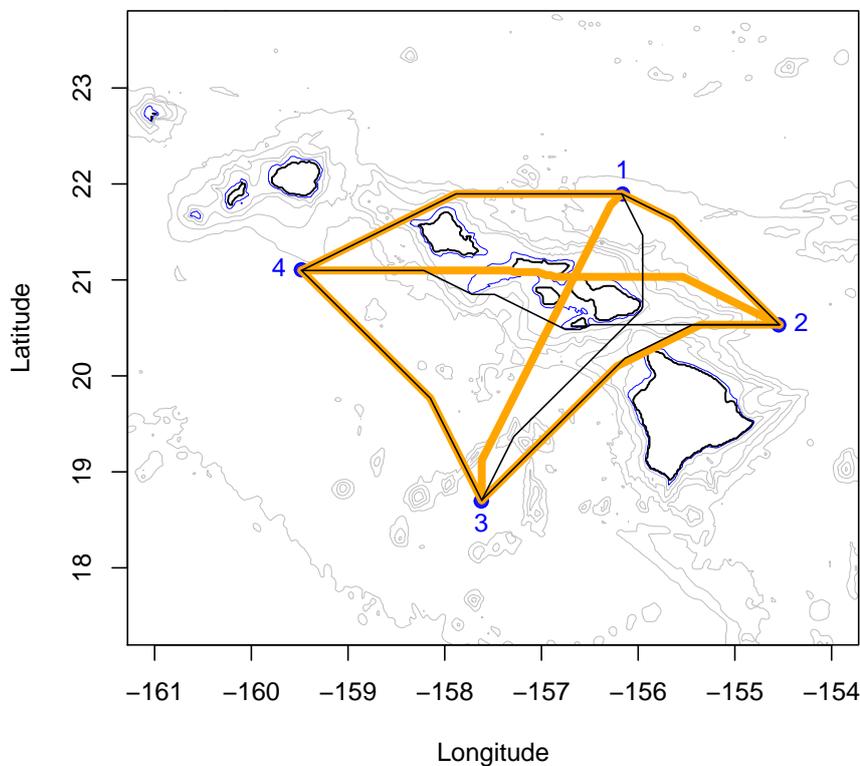

The argument `res` of `lc.dist()` controls whether path coordinates or distances between points (in kilometers) are outputted. Let's see how these different scenarios (no constraint: great-circle distance, `dist0` ; avoid landmasses: `dist1` ; avoid areas shallower than 200 m: `dist2`) affect distances between sampling points:

```
library(fossil)
dist0 <- round(earth.dist(sites), 0)
dist1 <- lc.dist(trans1, sites, res = "dist")
dist2 <- lc.dist(trans2, sites, res = "dist")
```

10

```
dist0

     1   2   3
2 226
3 387 381
4 355 517 331

dist1

     1   2   3
2 230
3 391 401
4 365 529 334

dist2

     1   2   3
2 230
3 423 403
4 365 533 334
```

Note: You can check out the help file for `lc.dist()` to see how we can combine these functions with cross-section calculations and plotting.

2.2 Landscape Genetics

The distance objects created in the section above are formatted as matrices that can be used in `R` or exported to be used in GenePop [15], TESS [7], or other software. As an example, these distances can be used to perform a Mantel test, as implemented in the package `ade4` (`mantel.rtest()` function ; [3, 5, 6]). The matrices produced in `marmap` are ready for use with `ade4`. For export and use in external programs, the function `write.matrix()` of the `MASS` package [16] or `write.table()` of the `utils` package will be helpful.

2.3 Shortest Great Circle Distances between points and isobath

Two functions of `marmap` allow for computing and plotting the shortest path following a great circle distance between a set of points on a map and an arbitrary isobath line. The function `dist2isobath()` depends on functions from packages `sp` [14] and `geosphere` [9] to compute the distances. By default (`isobath = 0`), the nearest location along the coastline is computed for each point.

```
# Load NW Atlantic xyz data and convert to class bathy
data(nw.atlantic)
atl <- as.bathy(nw.atlantic)

# Create vectors of latitude and longitude for 5 points
lon <- c(-70, -65, -63, -55, -48)
```

```
lat <- c(33, 35, 40, 37, 33)

# Compute distances between each point and the nearest location
# along the coastline
d <- dist2isobath(atl, lon, lat, isobath = 0)
d

    distance start.lon start.lat    end.lon  end.lat
1   487471.1       -70        33 -64.87297 32.26667
2   297613.8       -65        35 -64.73333 32.33571
3   435593.1       -63        40 -65.43333 43.46667
4   875011.1       -55        37 -59.73333 43.99167
5  1566914.4       -48        33 -64.71944 32.33333
```

We can then plot the bathymetry, add the 5 points, and plot the great circle lines to the nearest points on the coast using the function `linesGC()`:
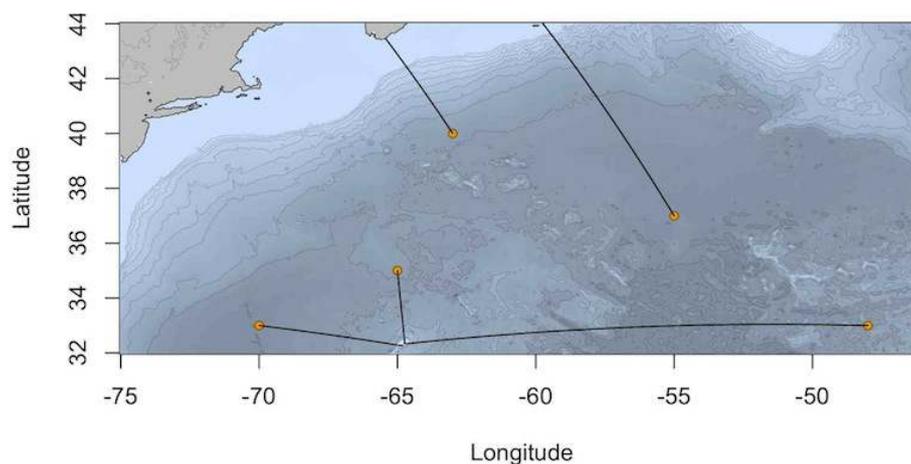
```
# Plot the bathymetry
plot(atl, image = TRUE, lwd = 0.1, land = TRUE,
     bpal = list(c(0, max(atl), "grey"), c(min(atl), 0, blues)))

# Make the coastline more visible
plot(atl, deep = 0, shallow = 0, step = 0, lwd = 0.6, add = TRUE)

# Add the 5 points
points(lon, lat, pch = 21, bg = "orange2", cex = 0.8)

# Add great circle lines
linesGC(d[, 2:3], d[, 4:5])
```


The same process can be used to compute and visualize the shortest great circle distance between a set of points and any arbitrary isoline of depth or altitude by setting the `isobath` argument of `dist2isobath()` to non-zero values (the chosen value must be within the range of altitude/depth for the region used to compute the distances).

3   3D plotting

R contains tools to plot data in three dimensions. We can use the function
`wireframe()` of the package `lattice` [4] to make a 3D representation of the
NW Atlantic and its seamount chains. `wireframe()` is not part of `marmap`, and
was therefore not meant to work with objects of class bathy. We need to use
the function `unclass()` to make our data available to `wireframe()`. Make sure
to adjust the `aspect` option of `wireframe()`, to minimize vertical exaggeration
and biased latitude / longitude aspect ratio.

```r
# Load NW Atlantic xyz data and convert to class bathy
data(nw.atlantic)
atl <- as.bathy(nw.atlantic)

library(lattice)
wireframe(unclass(atl), shade = TRUE, aspect = c(1/2, 0.1))
```
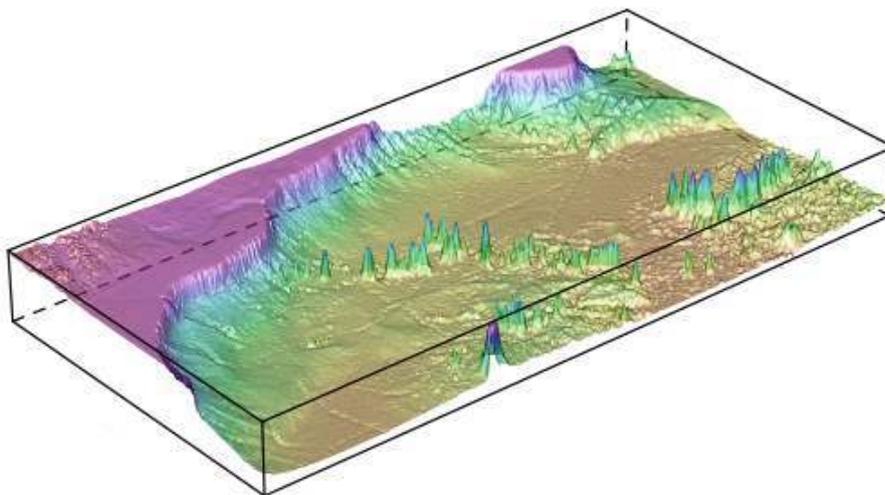

The `marmap` function `get.box()` can be coupled with the `lattice` function
`wireframe()` to produce 3D plots of belt transects of given width. Let's use the
NW Atlantic data to investigate these functions, and look at the New England
and Corner Rise seamount chains.

```r
data(nw.atlantic)
atl <- as.bathy(nw.atlantic)

plot(atl, xlim = c(-70, -52),
     deep = c(-5000, 0), shallow = c(0, 0), step = c(1000, 0),
     col = c("lightgrey", "black"), lwd = c(0.8, 1),
     lty = c(1, 1), draw = c(FALSE, FALSE))

belt <- get.box(atl, x1 = -68.6, x2 = -53.7, y1 = 42.4, y2 = 32.5,
                width = 3, col = "red")
```
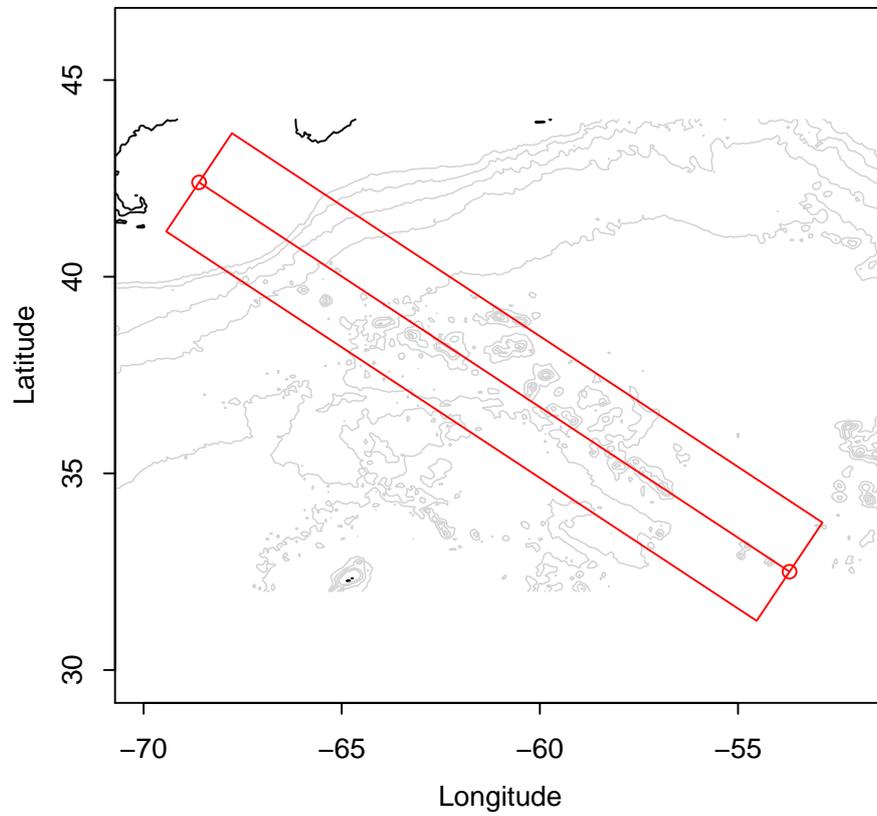
```
library(lattice)
wireframe(belt, shade = TRUE, zoom = 1.1,
          aspect = c(1/4, 0.1),
          screen = list(z = -60, x = -55),
          par.settings = list(axis.line = list(col = "transparent")),
          par.box = c(col = rgb(0, 0, 0, 0.1)))
```

row
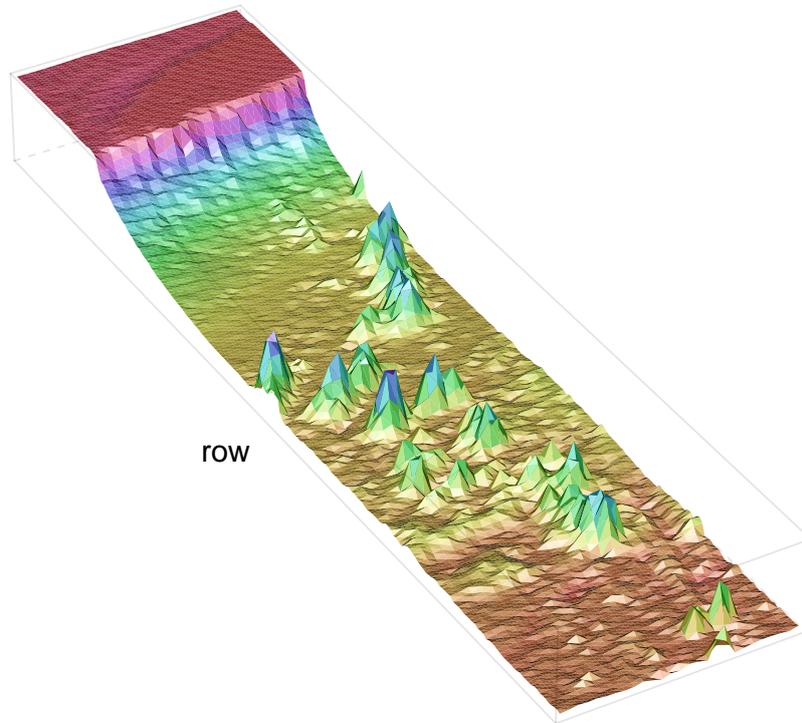
4   Working with big files

Data files containing bathymetry information can rapidly become huge (*e.g.* tens to hundreds of Mega-octets, millions of latitude-longitude-depth/altitude triplets), especially for hi-resolution bathymetry data recorded over large areas. If `marmap` can usually import large xyz files[1] to create `bathy` objects using `read.bathy()`, working with such objects can be difficult (if not impossible) depending on the amount of RAM available on your computer. More specifically, ressource-intensive tasks such as computing least cost paths might be extremely time consumming with datasets of millions of points. Even plotting with `plot.bathy()` can be very slow when too many countour lines are used, or when `image` is set to `TRUE`. In such situations, it is very useful to subset a big `bathy` object by either:

1. selecting a smaller region of a large `bathy` object while conserving its full resolution

2. lowering the resolution of the `bathy` object over the whole area

---

[1]The netcdf format is especially useful when dealing with big bathymetric files. Importing netcdf files to work with `marmap` is discussed in the `marmap-ImportExport` vignette.

3. a combination of the first 2 options, *i.e.* decreasing the resolution of the `bathy` object and selecting a smaller area for plotting or for other ressource-intensive computations.

For option 1, you can either use `get.box()` (see above), or `subsetBathy()` to select a smaller area of a large `bathy` object. `subsetBathy()` allows the selection of a non-rectangular area within a large `bathy` object to create a new, smaller `bathy` object of the same resolution. This function also has an interactive mode so that you can select an area of interest by clicking on a map.

For option 2, there is no built-in solution in `marmap`. However, it is pretty straightforward to decrease the resolution of a `bathy` object since it is just a `matrix` with a special `class`. If you have a big `bathy` object called `dat`, here is a solution:

```
# Derease the resolution of dat by a factor n
n <- 2
dat.lowres <- dat[seq(1, nrow(dat), by = n),
                  seq(1, ncol(dat), by = n)]

# Specify the class of the new object
class(dat.lowres) <- "bathy"
```

`dat.lowres` is now a new `bathy` object with a resolution 2 times lower than it was for `dat`.

Option 3 is just a combination of the 2 previous methods: first, create a `dat.lowres` object, then use `get.box()` or `subsetBathy()` to extract a smaller region out of it.

5 Interactions with other packages, projections

`marmap` interacts with multiple existing `R` packages for visualization and analysis, such as `lattice` for building three-dimensional plots, and `gdistance` for least-cost path calculations (see above). `marmap` also contains functions to ease interactions with other packages dedicated to the analysis of spatial data. Data of class `bathy` can be transformed into `RasterLayer` objets for use in the `raster` package [10] or into `SpatialGridDataFrame` objects for use in the packages `sp` [2, 14]. The full range of spatial analyses implemented in packages taking advantage of these classes are thus available for bathymetric data. The simple examples presented below illustrate how to apply an arbitrary projection to `bathy` objects using the function `projectRaster()` from the `raster` package (n.b. a working installation of the `rgdal` package is needed to use this function).

```
library(raster)

# Loads data of class bathy
data(hawaii)
```

```
# Creates an object of class raster
r1 <- marmap::as.raster(hawaii)

# Defines the target projection
newproj <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100
            +ellps=WGS84"

# Creates a new projected raster object
r2 <- projectRaster(r1, crs = newproj)

# Switches back to a bathy object
hawaii.projected <- as.bathy(r2)

# Plots both the original and projected bathy objects
plot(hawaii, image = TRUE, lwd = 0.3)
plot(hawaii.projected, image = TRUE, lwd = 0.3,
     xlab = "", ylab = "", axes = FALSE)
```
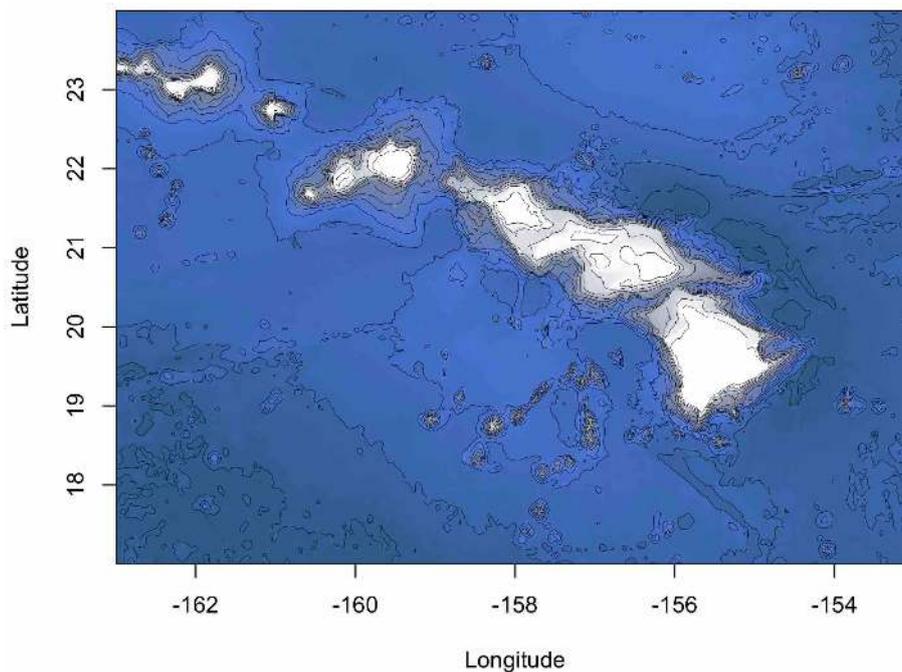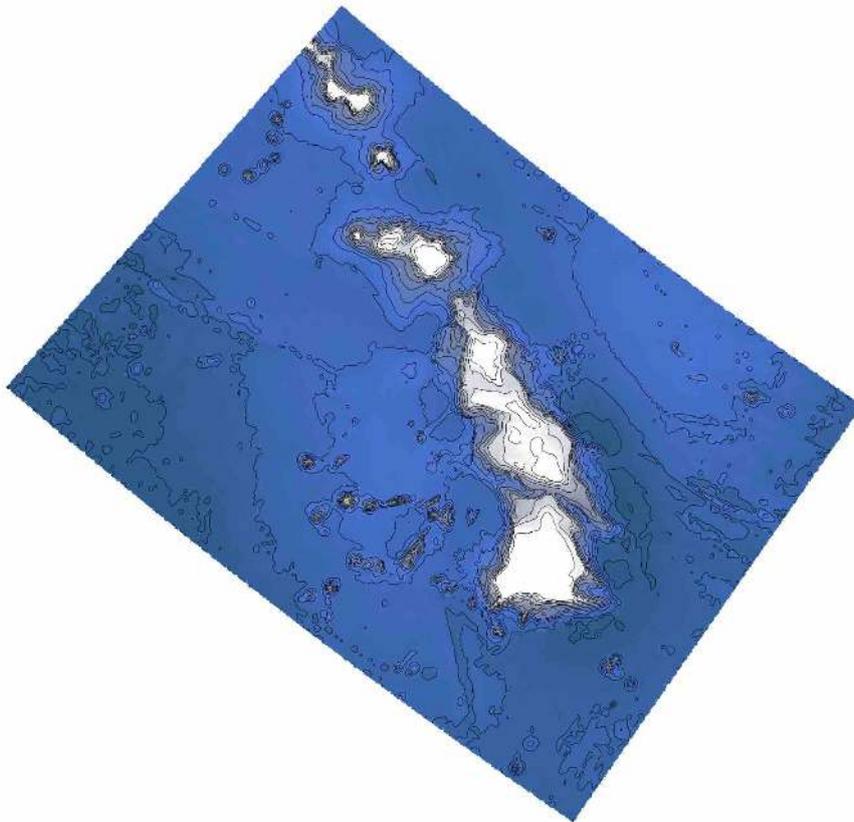
Here is another example for an orthographic projection of the whole world:

```
library(raster)

# Get data for the whole world. Careful: ca. 21 Mo!
world <- getNOAA.bathy(-180, 180, -90, 90, res = 15, keep = TRUE)

# Switch to raster
world.ras <- marmap::as.raster(world)

# Set the projection and project
my.proj <-   "+proj=ortho +lat_0=0 +lon_0=50 +x_0=0 +y_0=0"
world.ras.proj <- projectRaster(world.ras,crs = my.proj)

# Switch back to a bathy object
world.proj <- as.bathy(world.ras.proj)

# Set colors for oceans and land masses
blues <- c("lightsteelblue4", "lightsteelblue3",
           "lightsteelblue2", "lightsteelblue1")
greys <- c(grey(0.6), grey(0.93), grey(0.99))
```
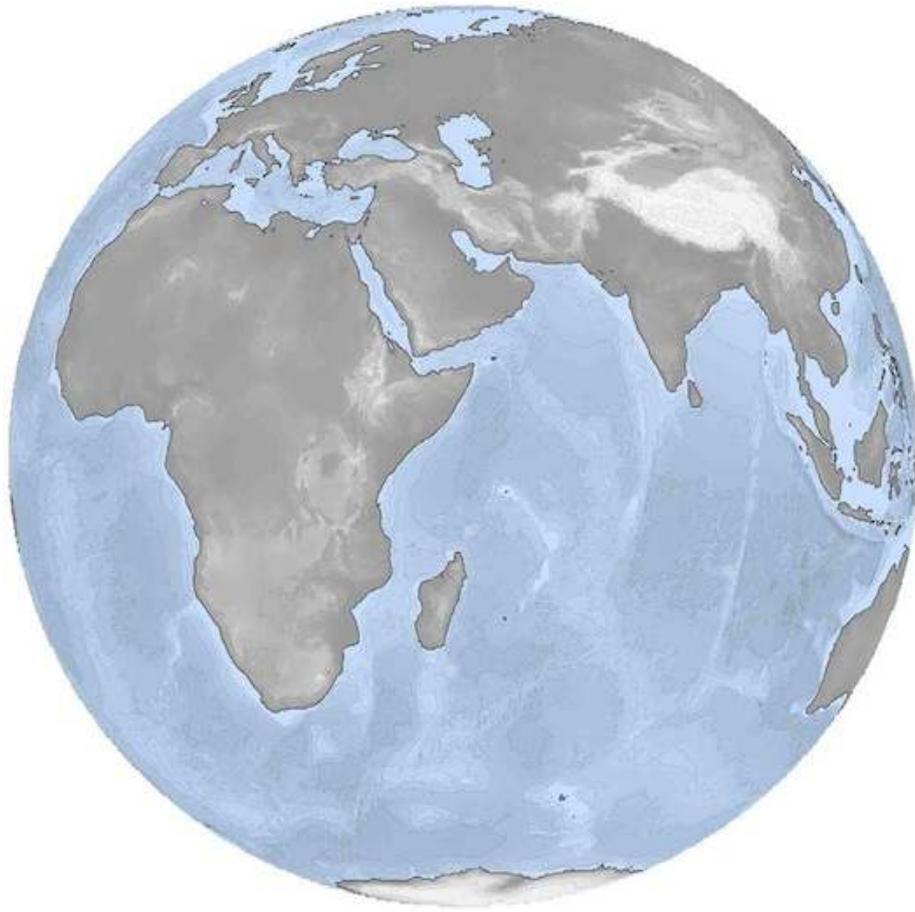
```
# And plot!
plot(world.proj, image = TRUE, land = TRUE, lwd = 0.05,
     bpal = list(c(0, max(world.proj, na.rm = T), greys),
                 c(min(world.proj, na.rm = T), 0, blues)),
     axes = FALSE, xlab = "", ylab = "")

plot(world.proj, n = 1, lwd = 0.4, add = TRUE)
```


A great list of available projections is available at http://www.remotesensing.
org/geotiff/proj_list/

References

[1] Amante C, Eakins BW (2009) Etopo1 1 arc-minute global relief model:
    Procedures, data sources and analysis. NOAA Technical Memorandum
    NESDIS NGDC-24: 1-19.

[2] Bivand RS, Pebesma EJ, Gomez-Rubio V (2013) Applied spatial data anal-
    ysis with R, Second edition. Springer, NY.

[3] Chessel D, Dufour A, Thioulouse J (2004) The ade4 package -I- One-table methods. R News 4: 5-10.

[4] Deepayan S, (2008) Lattice: Multivariate Data Visualization with R. Springer, New York.

[5] Dray S, Dufour A, Chessel D (2007) The ade4 package-II: Two-table and K-table methods. R News 7: 47-52.

[6] Dray S, Dufour A (2007) The ade4 package: implementing the duality diagram for ecologists. Journal of Statistical Software 22: 1-20.

[7] Durand E, Jay F, Gaggiotti OE, François O (2009) Spatial inference of admixture proportions and secondary contact zones. Molecular Biology and Evolution 26: 1963-1973.

[8] van Etten J (2014) gdistance: distances and routes on geographical grids. URL http://CRAN.R-project.org/package=gdistance. R package version 1.1-5.

[9] Hijmans RJ (2014) geosphere: Spherical Trigonometry. URL http://CRAN.R-project.org/package=geosphere. R package version 1.3-11.

[10] Hijmans RJ (2014) raster: Geographic data analysis and modeling. URL http://CRAN.R-project.org/package=raster. R package version 2.3-0.

[11] James DA, Falcon S (2013) RSQLite: SQLite interface for R. URL http://CRAN.R-project.org/package=RSQLite. R package version 0.11.4.

[12] NOAA National Geophysical Data Center. GEODAS Grid Translator - Design a grid. URL http://www.ngdc.noaa.gov/mgg/gdas/gd_designagrid.html.

[13] Pante E, Simon-Bouhet B (2013) marmap: A Package for Importing, Plotting and Analyzing Bathymetric and Topographic Data in R. PLoS ONE 8:e73051

[14] Pebesma EJ, Bivand RS (2005) Classes and methods for spatial data in R. R News. 5:9-13.

[15] Rousset F, (2008) GENEPOP'007: a complete re-implementation of the genepop software for Windows and Linux. Molecular Ecology Resources 8: 103-106.

[16] Venables WN, Ripley BD (2002) Modern Applied Statistics with S. Fourth edition. Springer, NY.