Making and using bathymetric maps in R with marmap

Eric Pante & Benoit Simon Bouhet

June 17, 2014

Contents

1 Introduction

In this vignette we introduce `marmap`, a package designed for manipulating bathymetric data in R. `marmap` uses simple latitude-longitude-depth data in ascii format and takes advantage of the advanced plotting tools available in R to build publication-quality bathymetric maps. Functions to query data (bathymetry, sampling information...) directly by clicking on `marmap` maps are available. Bathymetric and topographic data can also be used to constrain the calculation of realistic shortest path distances. Such information can be used in molecular ecology, for example, to evaluate genetic isolation by distance in a spatially-explicit framework.

2 A quick tutorial

In this section, we will produce bathymetric maps of Papua New Guinea, Hawaii and the NW Atlantic.

2.1 Getting data into R

Launch R. Navigate to your work folder (for example, with `setwd()`). Then launch the marmap package. The simplest way to get bathymetric data into R for use with `marmap` is to use the `getNOAA.bathy()` function. It queries the ETOPO1 dataset (Armante and Eakins 2009) hosted on the NOAA server, based on coordinates and a resolution given by the user (please note that this function depends on the availability of the NOAA server!). In one line, we can get the data into R and start plotting:

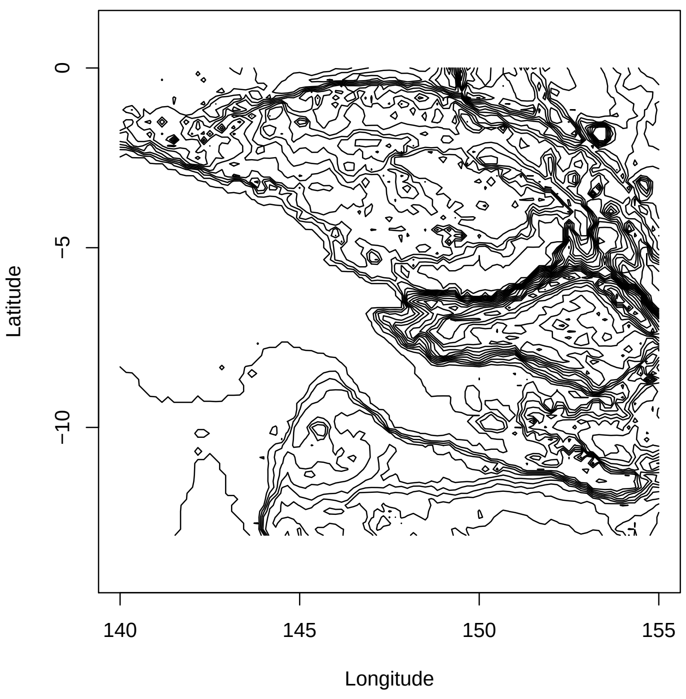
```
> library(marmap)
> getNOAA.bathy(lon1 = 140, lon2 = 155, lat1 = -13, lat2 = 0,
          resolution = 10) -> papoue
> summary(papoue)

Bathymetric data of class 'bathy', with 91 rows and 79 columns
Latitudinal range: -13 to 0 (13 S to 0 N)
Longitudinal range: 140 to 155 (140 E to 155 E)
Cell size: 10 minute(s)

Depth statistics:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -8750   -3123   -1540   -1641      -4    3711

First 5 columns and rows of the bathymetric matrix:
           -13 -12.833333 -12.666667 -12.5 -12.333333
140        -36        -35        -35   -35        -35
140.166667 -35        -34        -34   -34        -33
140.333333 -33        -32        -32   -32        -31
140.5      -30        -30        -30   -29        -29
140.666667 -28        -28        -27   -27        -27
```

`summary.bathy()` helps you check the data ; because bathy is a class, and R an object-oriented language, you just have to use `summary()`. R will recognize that you are feeding `summary()` an object of class bathy. This is also true for `plot.bathy` and `plot()`.

2.2 Plotting bathymetric data

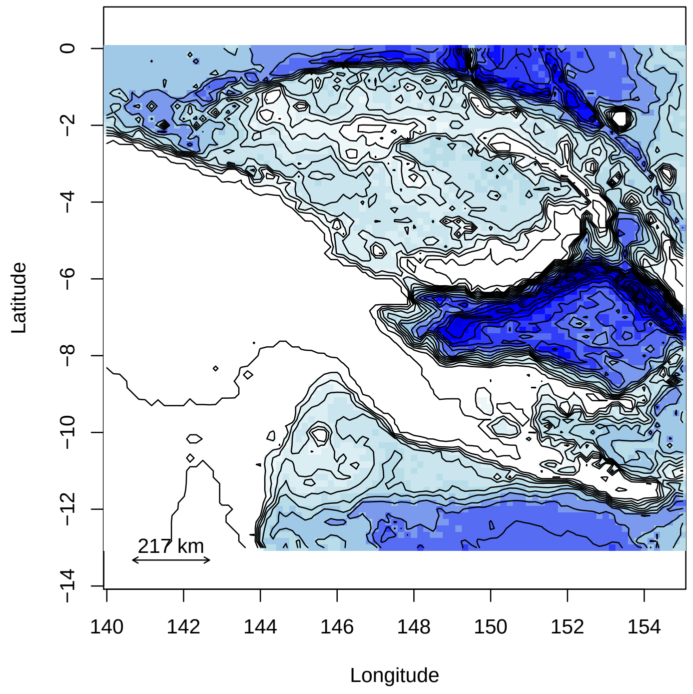
We can now use `plot.bathy()` (or `plot()`, because R will recognize the object is of class bathy) to map the data. You can see that the 10 minute resolution is a bit rough, but enough to demonstrate how `marmap` works (to increase the resolution, simply change the value for the `resolution` argument to a smaller value).

```
> plot(papoue)
```



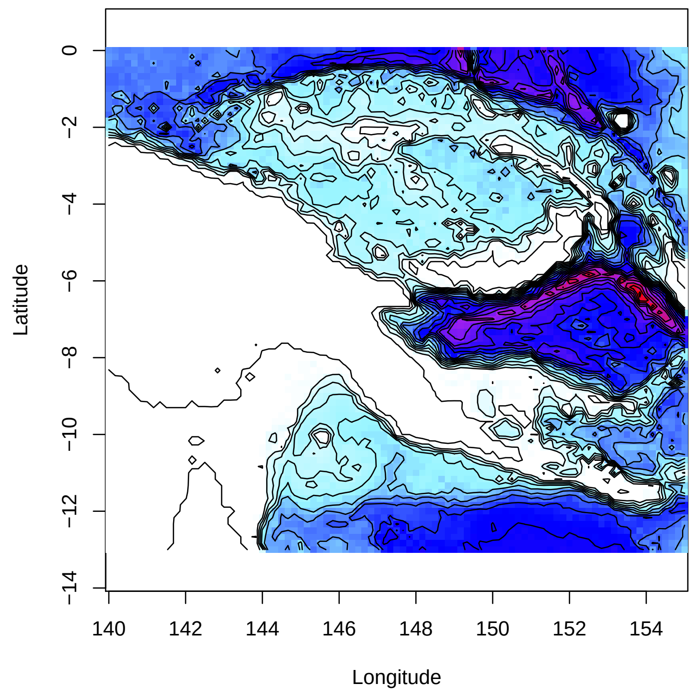
We can now use some of the options of `plot.bathy()` to make the map more informative. First, we can plot a heat map, using the built in color palette. We can also add a scale in kilometers.

```
> plot(papoue, image = TRUE)
> scaleBathy(papoue, deg = 2, x = "bottomleft", inset = 5)
```



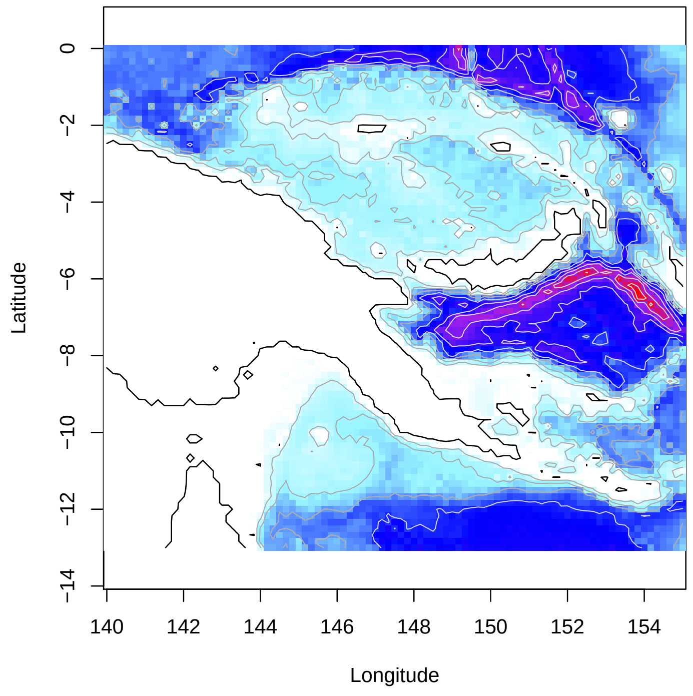
The `bpal` options allows you to use a custom color palette, which can be easily prepared with the R function `colorRampPalette()`. We store the color ramp in the object called `blues`, and when we call it in `plot.bathy()`, we specify how many colors need to be used in the palette (here 100).

```
> colorRampPalette(c("red","purple","blue","cadetblue1",
          "white")) -> blues
> plot(papoue, image = TRUE, bpal = blues(100))
```



For maps using the `image` option of `plot.bathy()`, you might see that the PDF rendering of your map is slightly different from the way it looks in R: the small space between cells becomes visible. This is probably due to the way your system handles PDFs. A simple way around this phenomenon is to export the map in a raster (rather than vector) format. You can use the `tiff()`, `jpeg()`, `bmp()` or `png()` functions available in R. This map looks a little crowded ; let's dim the isobaths (dark grey color and lighter line width), and strengthen the coastline (black color and thicker line width). The deepest isobaths will be hard to see on a dark blue background ; we can therefore choose to plot these in light grey to improve contrast. The option `drawlabel` controls whether isobath labels (e.g. "-3000") are plotted or not.

```
> plot(papoue, image = TRUE, bpal = blues(100),
           deep = c(-9000, -3000, 0), shallow = c(-3000, -10, 0),
           step = c(1000, 1000, 0), lwd = c(0.8, 0.8, 1),
           col = c("lightgrey", "darkgrey", "black"),
           lty = c(1, 1, 1), drawlabel = c(FALSE, FALSE, FALSE))
```



2.3 Using bathymetric data for further analysis

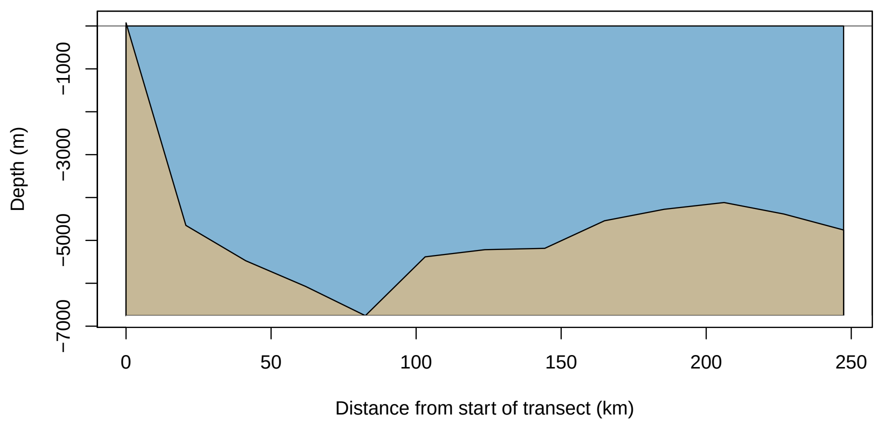
We can use the `get.transect()` and `plotProfile()` functions to extract and plot a depth cross section from the `papoue` dataset. `get.transect()` will use the coordinates you input to calculate the coordinates and depths along your transect, and calculate the great circle distance separating each point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

```
> get.transect(papoue, 151, -6, 153, -7, distance = TRUE)
```

```
        lon       lat   dist.km depth
1  151.0000 -6.000000   0.00000    73
2  151.1667 -6.083333  20.62796 -4650
3  151.3333 -6.166667  41.25328 -5474
4  151.5000 -6.250000  61.87610 -6072
5  151.6667 -6.333333  82.49630 -6755
6  151.8333 -6.416667 103.11374 -5383
7  152.0000 -6.500000 123.72859 -5216
8  152.1667 -6.583333 144.34070 -5185
9  152.3333 -6.666667 164.94996 -4542
10 152.5000 -6.750000 185.55650 -4274
11 152.6667 -6.833333 206.16021 -4117
12 152.8333 -6.916667 226.76095 -4385
13 153.0000 -7.000000 247.35888 -4757
```

We can plot that information on a map and make a cross section plot with `plotProfile()`. Again, the very low resolution of the dataset produces an analysis with little information. You can get transect information and make a cross-section plot by directly clicking on the map, using the `locator` option of `get.transect()`.

```
> get.transect(papoue, 151, -6, 153, -7,
          distance = TRUE) -> transect
> plotProfile(transect)
```



We can also use `get.depth()` to retrieve depth information by either clicking on the map or by providing a set of longitude/latitude pairs (see help pages). 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:

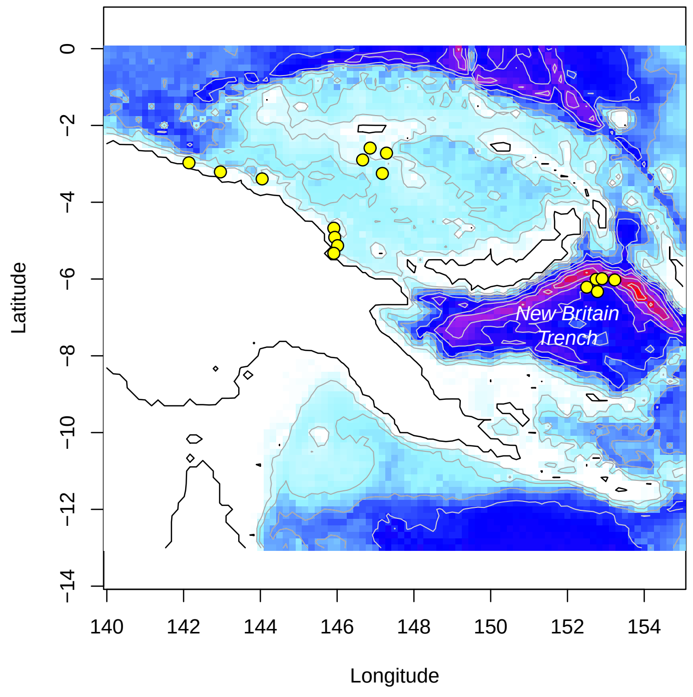
```
> get.depth(papoue, distance=TRUE)
Waiting for interactive input: click any number of times on the map, then press 'Esc'
       Lon       Lat Depth  Dist.km
1 146.0200 -2.601702  -758   0.0000
2 147.6167 -1.844152  -583 196.3933
3 149.3193 -2.607345 -2121 366.4942
4 150.7295 -4.249027 -2289 553.8867
```

`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, 153.2314)
> 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, -6.023344)
> paste("station",1:16, sep = "") -> station
> data.frame(x, y, station) -> sampling
```

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, bpal = blues(100),
          deep = c(-9000, -3000, 0), shallow = c(-3000, -10, 0),
          step = c(1000, 1000, 0), lwd = c(0.8, 0.8, 1),
          col = c("lightgrey", "darkgrey", "black"),
          lty = c(1, 1, 1), drawlabel = c(FALSE, FALSE, FALSE))
> # add points from the sampling.csv, 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)
```

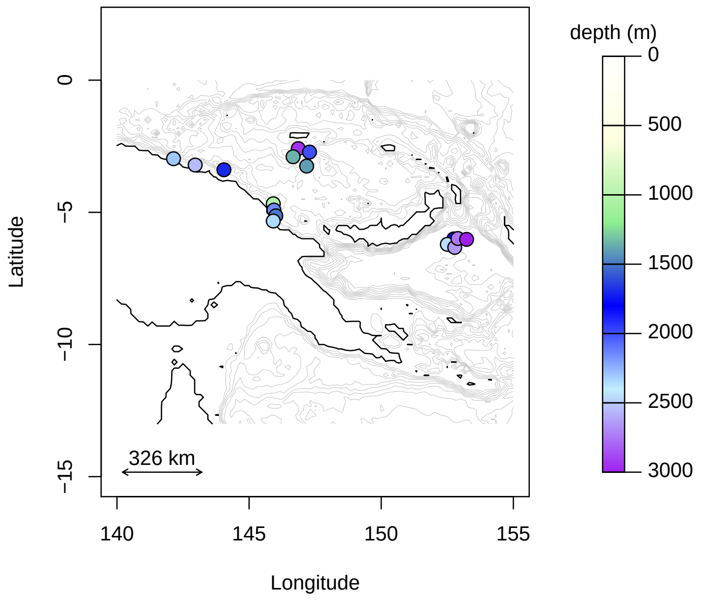


By clicking on the map, we can select the area in the New Britain Trench, to get information on the sampling stations of that area. `get.sample()` will detect that there are samples in the area selected, and return the locations relative for these samples.

```
> # 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
```

We can use the depth data when plotting points:

```
> # make a table of fake sampling information, with fake depth
> samp.depth = sample(seq(-3000, -1000, by = 50), size = 16)
> data.frame(sampling$x, sampling$y, samp.depth) -> sp
> names(sp) <- c("lon", "lat", "depth")
> head(sp)

       lon       lat depth
1 142.1390 -2.972065 -2300
2 142.9593 -3.209449 -2550
3 144.0466 -3.391399 -1700
4 145.9141 -4.675720 -1000
5 145.9372 -4.914153 -2150
6 146.0115 -5.130116 -1500

> # plot map
> par(mai=c(1,1,1,1.5))
> plot(papoue, deep = c(-4500, 0), shallow = c(-50, 0), step = c(500, 0),
          lwd = c(0.3, 1), lty = c(1, 1), col = c("grey", "black"),
          drawlabels = c(FALSE, FALSE))
> scaleBathy(papoue, deg = 3, x = "bottomleft", inset = 5)
> # set color palette
> max(-sp$depth, na.rm = TRUE) -> mx
> colorRampPalette(c("white", "lightyellow", "lightgreen",
          "blue", "lightblue1", "purple")) -> ramp
> blues <- ramp(mx)
> # plot points and color depth scale
> points(sp[,1:2], col = "black", bg = blues[-sp$depth],
          pch = 21, cex = 1.5)
> library(shape)
> colorlegend(zlim = c(mx, 0), col = rev(blues), main = "depth (m)",
          posx = c(0.85, 0.88))
```

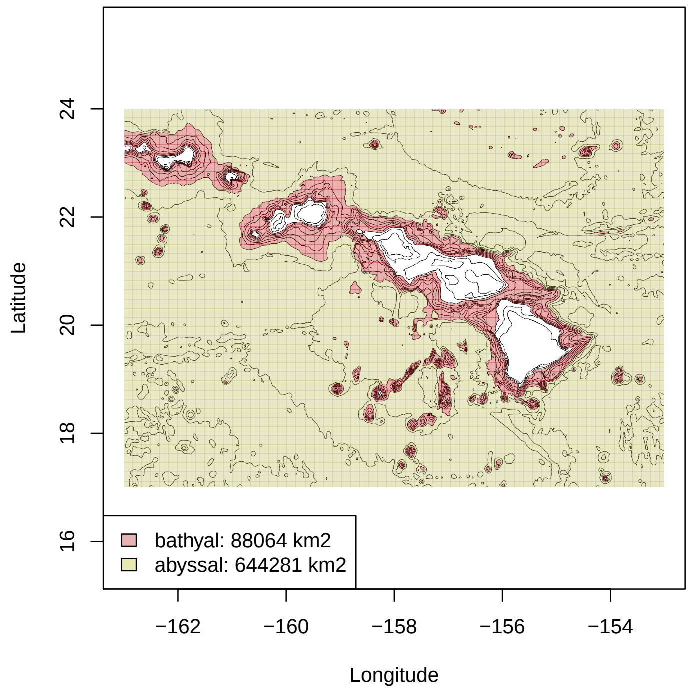


The function `get.area()` can be used to calculate the projected surface area (the projecting surface being the ocean surface). For example, in the case of the Hawaiian Archipelago, we can calculate the surface area of the bathyal (1,000 to 4,000 m) and abyssal regions (4,000 to about 6,000 m).

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

The function `get.area()` returns a surface area in square kilometers (`$Square.Km`), and a matrix of zeros and ones delimiting the area of interest. The `$Lon`, `$Lat` and `$Area` objects can be used to display these areas:

```
> plot(hawaii, lwd = 0.2)
> image(bathyal$Lon, bathyal$Lat, bathyal$Area,
          col = c("transparent", rgb(0.7, 0, 0, 0.3)), add = TRUE)
> image(abyssal$Lon, abyssal$Lat, abyssal$Area,
          col = c("transparent", rgb(0.7, 0.7, 0.3, 0.3)), add = TRUE)
> legend("bottomleft",
          legend = c(paste("bathyal:", ba, "km2"),
          paste("abyssal:", ab, "km2")),
          fill = c(rgb(0.7, 0, 0, 0.3), rgb(0.7, 0.7, 0, 0.3)))
```



2.4 Using bathymetric data for least-cost path analysis

`marmap` contains functions to facilitate least-cost path analysis that are based on the `raster` and `gdistance` packages (van Etten 2012a, 2012b). `gdistance` calculates routes 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 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 `min.depth` and `max.depth` ensure that the path computed by `dist.geo` 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.

`trans1` is a transition object contained 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.

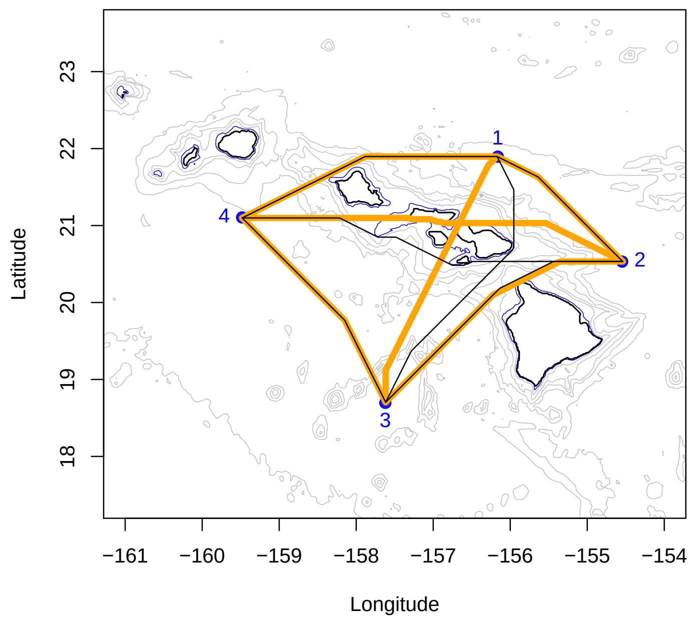
```
> trans1 <- trans.mat(hawaii)
> trans2 <- trans.mat(hawaii, min.depth = -200)
```

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 option `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) effect 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")
> 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.

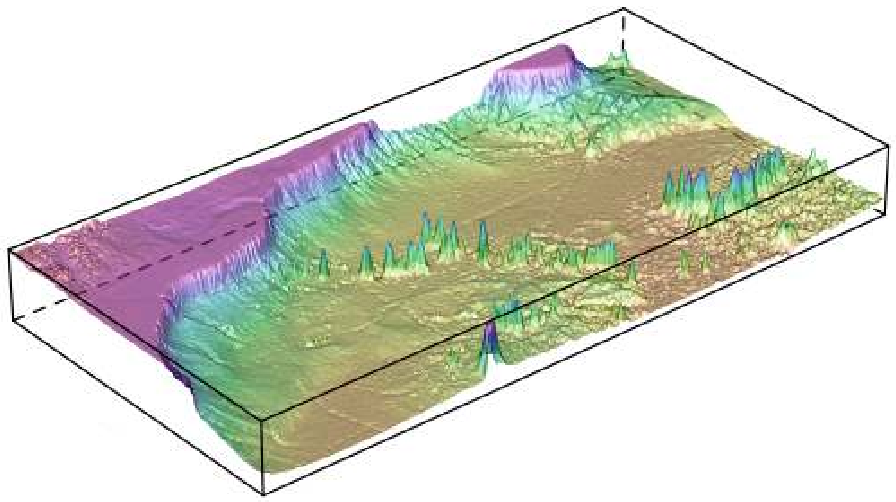
2.5 Landscape Genetics

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 (Rousset 2008), TESS (Durand et al 2009), 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 ; Chessel and Dufour 2004, Dray et al 2007, Dray and Dufour 2007). 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 (Venables and Ripley 2002) will be helpful.

2.6 3D plotting

R contains tools to plot data in three dimensions. We can use the function `wireframe()` of the package `lattice` 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.

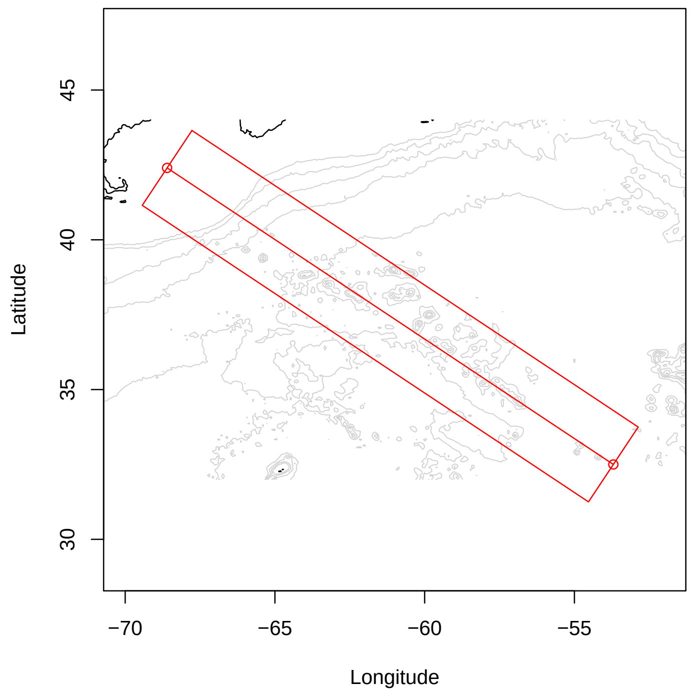
```
data(nw.atlantic)
atl <- as.bathy(nw.atlantic)
library(lattice)
wireframe(unclass(atl), shade = TRUE, aspect = c(1/2, 0.1))
```



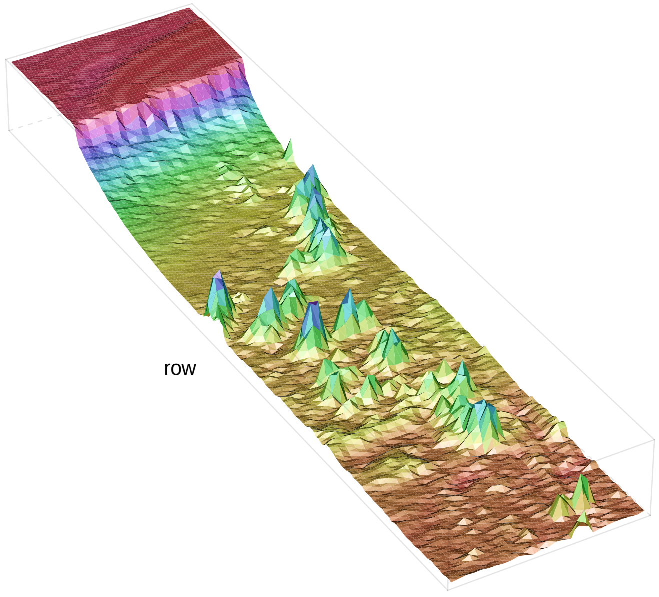
The `marmap` function `get.box()` can be coupled with the `lattice` function `wireframe` to produce 3D plots of belt transects of given width. Let's use the

NW Atlantic data to investigate these functions, and look at the New England and Corner Rise seamount chains.

```
> 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))
> get.box(atl, x1 = -68.6, x2 = -53.7, y1 = 42.4, y2 = 32.5,
          width = 3, col = "red") -> out
```



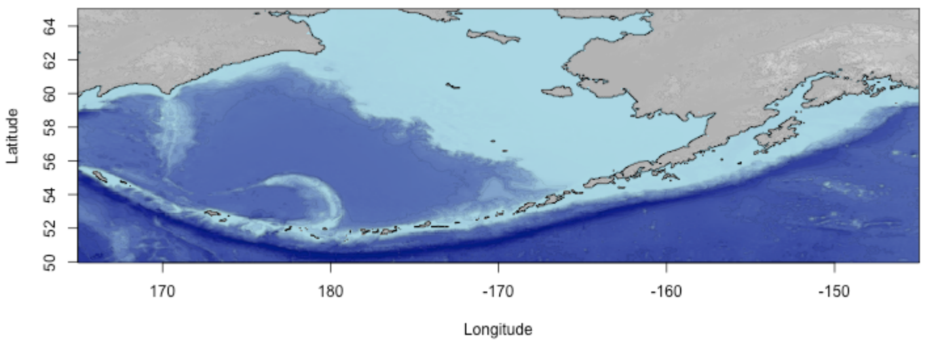
```
> library(lattice)
> wireframe(out, shade = TRUE, zoom = 1.1,
          aspect = c(1/4, 0.1),
          screen = list(z = -60, x = -55),
          par.settings = list(axis.line = list(col = "transparent")),
          par.box = c(col = rgb(0, 0, 0, 0.1)))
```



2.7 Preparing maps in the Pacific antimeridian region

The antimeridian (or antemeridian) is the 180th meridian and is located about in the middle of the Pacific Ocean, east of New Zealand and Fidji, west of Hawaii and Tonga. If you want to prepare a map of the Aleutian Islands (Alaska), your latitude values may, for example, go from 165 to 180 degrees East, and 180 to 165 degrees West. Crossing the antemeridian means that you will need to download data for the eastern (165 to 180) and the western (-180 to -165) portions of the area of interest (for example, GEBCO will tell you "The Westernmost is more Easterly than the Easternmost. Please amend your search query" if you try to download data for the Aleutians in one step). `getNOAA()` has an argument to deal with the antemeridian region. For the Aleutians, you would use the `antimeridian` argument. `summary.bathy()` can interpret antimeridian areas as well. When you plot your antimeridian region, the default behavior of `plot.bathy()` is to scale longitudes from 0 to 360 degrees (170E to 170W would be displayed as 170, 190 instead of 170, -170). You can use the argument `axes=FALSE` in `plot.bathy()` and add correct labels with `antimeridian.box()`. We have set the default behavior of `plot.bathy()` in this way to remind the user that the scale of the bathy object, in the antimeridian region, goes from 0 to 360; if you need to plot points on the map, you need to take this into account (i.e. a point at -170 longitude must be plotted using 190, not 170 or -170).

```
> getNOAA.bathy(165,-145,50,65, resolution=5,
  + antimeridian=TRUE) -> aleu
> summary(aleu)
> plot(aleu, image=TRUE,
  + bpal=list(c(0,max(aleutians),grey(.7),grey(.9),grey(.95)),
  + c(min(aleutians),0,"darkblue","lightblue")),
  + land=TRUE,lwd=0.1,axes=FALSE)
> plot(aleutians,n=1,lwd=.8,add=T)
> antimeridian.box(aleu)
```



Alternatively, it is possible to import two compatible `bathy` objects (for instance from GEBCO), one for the eastern part and one for the western part of the area of interest. The function `collate.bathy` takes care of the stitching process: relabelling longitudes in the 0-360 degrees range, removing duplicated data (i.e. the data for longitude 180 is often present once in each individual dataset and thus needs to be removed once), etc. Providing that we downloaded two files "east.nc" and "west.nc" from the GEBCO website, creating a proper bathy object for the antimeridian region is as simple as:

```
> a <- getGEBCO.bathy("east.nc")
> b <- getGEBCO.bathy("west.nc")
> stitched <- collate.bathy(a,b)
```

3 Data import and export strategies in marmap

3.1 Overview of the different import and export strategies available in marmap

`getNOAA.bathy()` is the easiest way to load data into R, but it depends on the NOAA download protocol, and one must have an internet connection (see above). However, setting the `keep` argument to `TRUE` will save on disk the data downloaded from the NOAA servers when the function is called for the first time. Any subsequent call to `getNOAA.bathy()` with the same list of arguments (i.e. same longitudes, latitudes and resolution) will preferentially load the dataset saved on disk in the current working directory. This allows the users to run scripts without having to query the NOAA servers and download the same data again and again, making the use of `getNOAA.bathy()` possible even off-line. `read.bathy()` allows import of data into R, and this data can be located on a drive ; an internet connection is therefore not mandatory. This is a good way to import data that have been saved locally on your drive, and may be faster than re-downloading data from the NOAA server at the beginning of each R session. If the user is building maps routinely, we propose two functions to create a local database that can be accessed from within R. These functions

Function	Job	Input	Output	Internet
`getNOAA.bathy()`	downloads data from NOAA server	coordinates of bounding box and resolution	data matrix of class bathy	yes
`readGEBCO.bathy()`	imports data from GEBCO file	name of external file in netCDF format	data matrix of class bathy	no
`read.bathy()`	imports data into R	name of external file with xyz data	data matrix of class bathy	no
`setSQL()`	creates a local SQL database of bathymetric data	name of external file with xyz data	an SQL database	no
`subsetSQL()`	queries a local SQL database	coordinates of bounding box and resolution	data matrix of class bathy	no
`as.xyz()`	converts a dataset of class bathy into an xyz table	dataset of class bathy (an R object)	an xyz table (an R object)	no
`as.bathy()`	converts an xyz table into an dataset of class bathy	an xyz table (an R object)	dataset of class bathy (an R object)	no

are `setSQL()` and `subsetSQL()`.

3.2 Importing bathymetric data from GEBCO: readGEBCO.bathy()

`readGEBCO.bathy()` provides a data source alternative to the NOAA-hosted ETOPO1 data. The GEBCO data, hosted on the British Oceanographic Data Center server, is available at the 30 second and 1 minute resolutions. Both types can be imported using `readGEBCO.bathy()`, using the `ncdf` package to load netCDF data into R. The argument `db` specifies whether data was downloaded from the 30 arcseconds database (GEBCO_08) or the 1 arcminute database (GEBCO_1min, the default). A third database type, GEBCO_08 SID, is available from the website. This database contains a Source IDentifier (SID) specifying which grid cells have depth information based on soundings; it does not contain bathymetry or topography data. `readGEBCO.bathy` can read this type of database with `db = "GEBCO_08"`, and only the SID information will be included in the object of class `bathy`. Therefore, to display a map with both the bathymetry and the SID information, you will have to download both datasets from GEBCO, and import and plot both independently. Here is an example for the region of the Mediterranean Sea including Corsica and Sardinia:

```
> readGEBCO.bathy("gebco_08_7_38_10_43_corsica.nc", db="GEBCO_08") -> med
> summary(med)      # the bathymetry data
> readGEBCO.bathy("gebco_SID_7_38_10_43_corsica.nc", db="GEBCO_08")-> sid
> summary(sid)     # the SID data

# a pretty custom color palette
> colorRampPalette(c("lightblue","cadetblue2","cadetblue1","white")) -> blues

# a first plot for bathymetry
> plot(med, n=1, im=T, bpal=blues(100),
      main="Corsica & Sardinia bathymetry\n GEODAS 08 & SID datasets")
# a second layer with the SID data
> contour(as.numeric(rownames(sid)), as.numeric(colnames(sid)), sid,
      drawlabels=F, lwd=.1, add=T)
```

The argument `resolution` specifies the resolution of the object of class `bathy`. Because the resolution of GEBCO data is rather fine, we offer the possibility of downsizing the dataset with `resolution`. `resolution` is in units of the selected database: in "GEBCO_1min", `resolution` is in minutes; in "GEBCO_08", `resolution` is in 30 arcseconds (that is, `resolution = 3` corresponds to 3x30sec, or 1.5 arcminute).

3.3 Getting bathymetric data from an xyz file: read.bathy()

`read.bathy()` will read xyz data from any source. Here, we will get ETOPO1 data hosted on the NOAA GEODAS server (NOAA National Geophysical Data Center 2013). To get the data, use the following link:

```
http://www.ngdc.noaa.gov/mgg/gdas/gd_designagrid.html
```

To prepare data from NOAA, give a name to your custom grid, choose the database (ETOPO1 1-minute Global Relief), fill the custom grid form (upper latitude: 0, lower latitude: 13S, left longitude: 140E, right longitude: 155E) for a grid cell size of 10 minute, and choose "XYZ (lon,lat,depth)" as the "Output Grid Format", "No Header" as the "Output Grid Header", and either of the space, tab of comma as the column delimiter (either can be used, but "comma" is the default import format of `read.bathy()`). Choose "omit empty grid cells" to reduce memory usage. Submit your job, and retrieved your data. You will get a zipped folder, in which you will find (in a subfolder) a .xyz file with your data. Place it, for example, in your work folder.

The resolution of 10 minutes is a low resolution that will keep the size of the example file small, about 200 kb. Increasing the resolution to 1 minute would result in a file size of about 20 mb.

Launch R. Navigate to your work folder (for example, with `setwd()`). Then launch the marmap package. and load your xyz data (we will call it "png.xyz") with `read.bathy()`. This converts your data into an R object of class "bathy." `summary.bathy()` helps you check the data ; because bathy is a class, and R an object-oriented language, you just have to use `summary()`, because R will recognize that you are feeding `summary()` an object of class bathy. This is also true for `plot.bathy` and `plot()`.

```
> library(marmap)
> read.bathy('png.xyz', header = FALSE, sep = "\t") -> papoue
> summary(papoue)
```

```
Bathymetric data of class 'bathy', with 91 rows and 79 columns
Latitudinal range: -13 to 0 (13 S to 0 N)
Longitudinal range: 140 to 155 (140 E to 155 E)
Cell size: 10 minute(s)

Depth statistics:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -8750   -3123   -1540   -1641      -4    3711

First 5 columns and rows of the bathymetric matrix:
           -13 -12.833333 -12.666667 -12.5 -12.333333
140        -36        -35        -35   -35        -35
140.166667 -35        -34        -34   -34        -33
140.333333 -33        -32        -32   -32        -31
140.5      -30        -30        -30   -29        -29
140.666667 -28        -28        -27   -27        -27
```

The `read.bathy` function can import bathymetric data for non rectangular areas as is often the case for custom datasets acquired by various types of sonar systems (e.g. Multibeam Echo Sounders). Please note however that depending on the size of the xyz file, the resolution of the data and the shape of the area covered by the data, the import can take up to sevral minutes.

3.4 Getting bathymetric data from NOAA: local SQL database

`setSQL()` and `subsetSQL()` create and query a local SQL database for bathymetric data. These tools are made for routine use with no internet connection. The full ETOPO1 database, or a subset (for example), can be downloaded on your computer, and used to set an SQL database, which size will be approximately the same as your original xyz data (unzipped ETOPO1 is about 5 Go). The advantage of SQL, a language for querying large databases, are manyfold. Its use will allow rapid upload of data into R, directly as bathy objects (and therefore directly useable for plotting and analysis) with a smaller footprint on your memory than if you tried to load a very large xyz file into R and then subset-ed it. Here is a simple example on how to set up and use an SQL database for marmap.

Use a local file with xyz data (we can re-use the png.xyz that we created above for use with `read.bathy()`), and submit it to `setSQL()`. Make sure that no file called `bathy_db` is present in your working directory. Also, make sure that the package `RSQLite` (James and Falcon 2012) is installed and properly working.

```
> require(RSQLite)
> setSQL(bathy = "png.xyz", sep = "\t")
```

```
[1] TRUE
```

This will created a file `bathy_db` in your directory, which size is about the size of (or larger than) your original data. If you want to create a database for frequent use, you just need to do this once. `subsetSQL()` will know where to get the data in future R sessions. If `setSQL()` worked properly, it will return `TRUE`. If there is a problem (e.g. database connection already open, database file already created ...) it will return `FALSE`. Lets query a subset of the png dataset, and check that it is indeed what we asked for with the `summary.bathy()` function:

```
> subsetSQL(min_lon = 145, max_lon = 150,
           min_lat = -2, max_lat = 0) -> test
> summary(test)
```

```
Bathymetric data of class 'bathy', with 29 rows and 11 columns
Latitudinal range: -1.83 to -0.17 (1.83 S to 0.17 S)
Longitudinal range: 145.17 to 149.83 (145.17 E to 149.83 E)
Cell size: 10 minute(s)

Depth statistics:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -6650   -3282   -2076   -2594   -1543      72

First 5 columns and rows of the bathymetric matrix:
           -1.833333 -1.666667  -1.5 -1.333333 -1.166667
145.166667     -1001     -1348  -249     -1774     -2079
145.333333     -1137     -1579 -1938     -1794     -1957
145.5          -1069     -1833 -2007     -2097     -2166
145.666667     -1295     -2020 -2123     -2301     -2289
145.833333     -1728     -1912 -1981     -2183     -2350
```

Finally, if you are done with the SQL dataset, you can remove it with

```
> system("rm bathy_db")
```

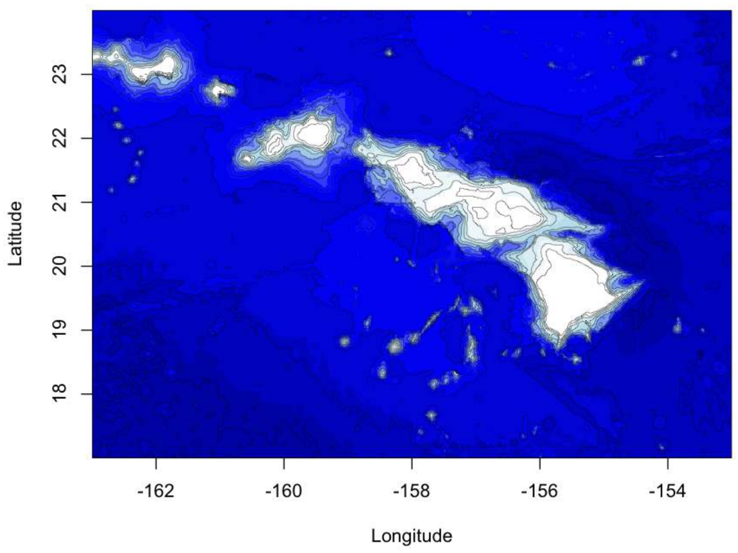
4 Miscellaneous

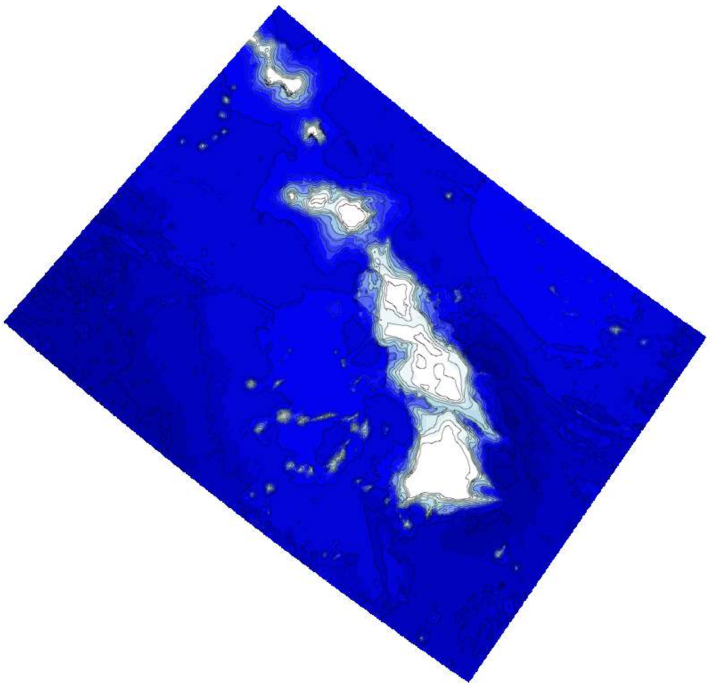
4.1 Interactions with other packages

`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 from class `bathy` can be transformed into `RasterLayer` objets for use in the `raster` package [7] or into `SpatialGridDataFrame` objects for use in the packages `sp` [2, 10]. The full range of spatial analyses implemented in packages taking advantage of these classes are thus available for bathymetric data. The simple example 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).

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

```
# Creates an object of class raster
> r1 <- 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)
```





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 (2008) Applied spatial data analysis with R. Springer, NY.

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

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

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

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

[7] van Etten RJHJ (2012) raster: Geographic data analysis and modeling. URL `http://CRAN.R-project.org/package=raster`. R package version 2.0-41.

[8] van Etten J (2012) gdistance: Distances and routes on geographical grids. URL `http://CRAN.R-project.org/package=gdistance`. R package version 1.1-4.

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

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

[11] James DA, Falcon S (2012) RSQLite: SQLite interface for R. URL http://CRAN.R-project.org/package=RSQLite. R package version 0.11.2.

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

[13] Venables, W. N. and B. D. Ripley. 2002. Modern Applied Statistics with S. Fourth edition. Springer.