Data import and export strategies in `marmap`

Eric Pante & Benoit Simon-Bouhet

November 3, 2014

Contents

---

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

Function	Job	Input	Output	Internet
`getNOAA.bathy()`	downloads data from NOAA servers	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 or an object of class `raster` into an dataset of class `bathy`	an xyz table (an `R` object)	dataset of class `bathy` (an `R` object)	no

2 Importing bathymetric data from GEBCO

`readGEBCO.bathy()` provides a data source alternative to the NOAA-hosted ETOPO1 data [1]. The GEBCO data, hosted on the British Oceanographic Data Center server (http://www.gebco.net), is available at the 30 second and 1 minute resolutions. Both types can be imported using `readGEBCO.bathy()`, using the `ncdf` package [5] 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. The function `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:

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

# the SID data
sid <- readGEBCO.bathy("gebco_SID_7_38_10_43_corsica.nc",
                       db = "GEBCO_08")
summary(sid)

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

# a first plot for bathymetry
plot(med, n = 1, image = TRUE, 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 = FALSE, lwd = 0.1, add = TRUE)
```

Because the resolution of GEBCO data is rather fine, we offer the possibility of downsizing the dataset with the `resolution` argument of `readGEBCO.bathy()`. This argument specifies the resolution of the object of class `bathy` the user gets after importing GEBCO data in `R`. `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 3 times 30 sec, or 1.5 arcminute resolution).

3 Other sources of netcdf files

One of the most widely used format for georeferenced data is netcdf. Bathymetric data is often presented as netcdf files since it is a compact, self-describing, machine-independent data format especially usefull for large scale and/or high resolution gridded data. For instance, the European Marine Observation and Data Network (emodnet: http://www.emodnet.eu/bathymetry) makes bathymetric data publicly available for most european waters. Among other formats, netcdf files of bathymetry for large regions (*e.g.* Bay of Biscay and Iberian coasts, Celtic Seas, Greater North Sea...) are available for download. `marmap` does not offer an automated way to import such files since almost every netcdf file is unique. However, a small number of very simple steps make it possible to import netcdf files and transform them to `bathy` objects. Here is an example with the `Celtic Seas.mnt` file downloaded on the emodnet website:

```
# Load relevant packages
library(marmap) ; library(ncdf)

# Load the netcdf file into R using the ncdf package
nc <- open.ncdf("Celtic Seas.mnt")

# Print the content of the file
nc

[1] "file Celtic Seas.mnt has 9 dimensions:"
[1] "CIB_BLOCK_DIM   Size: 1024"
[1] "mbHistoryRecNbr   Size: 20"
[1] "mbNameLength   Size: 20"
[1] "mbCommentLength   Size: 256"
[1] "mbLabelLength   Size: 40"
[1] "LAYERS_HEADERS   Size: 20"
[1] "LINES   Size: 3840"
[1] "COLUMNS   Size: 6120"
[1] "mbCDILength   Size: 100"
[1] "-------------------------"
[1] "file Celtic Seas.mnt has 17 variables:"
[1] "int mbHistDate[mbHistoryRecNbr]
          Longname:History date Missval:2147483647"
[1] "int mbHistTime[mbHistoryRecNbr]
          Longname:History time (UT) Missval:NA"
[1] "byte mbHistCode[mbHistoryRecNbr]
          Longname:History code Missval:0"
[1] "char mbHistAutor[mbNameLength,mbHistoryRecNbr]
          Longname:History autor Missval:NA"
[1] "char mbHistModule[mbNameLength,mbHistoryRecNbr]
          Longname:History module Missval:NA"
[1] "char mbHistComment[mbCommentLength,mbHistoryRecNbr]
          Longname:History comment Missval:NA"
[1] "char Layer_name[mbNameLength,LAYERS_HEADERS]
          Longname:Nom de la couche Missval:NA"
```

```
[1] "short DEPTH[COLUMNS,LINES]
          Longname:DEPTH Missval:32767"
[1] "short SMO_DEPTH[COLUMNS,LINES]
          Longname:SMO_DEPTH Missval:32767"
[1] "int OFF_DEPTH[COLUMNS,LINES]
          Longname:OFF_DEPTH Missval:2147483647"
[1] "int VSOUNDINGS[COLUMNS,LINES]
          Longname:VSOUNDINGS Missval:2147483647"
[1] "short MIN_SOUNDING[COLUMNS,LINES]
          Longname:MIN_SOUNDING Missval:32767"
[1] "short MAX_SOUNDING[COLUMNS,LINES]
          Longname:MAX_SOUNDING Missval:32767"
[1] "short STDEV[COLUMNS,LINES]
          Longname:STDEV Missval:32767"
[1] "char CDI[mbCDILength,COLUMNS,LINES]
          Longname:CDI Missval:NA"
[1] "byte CELL_NUMBER[COLUMNS,LINES]
          Longname:CELL_NUMBER Missval:127"
[1] "int CDI_SOUNDINGS[COLUMNS,LINES]
          Longname:CDI_SOUNDINGS Missval:2147483647"
```

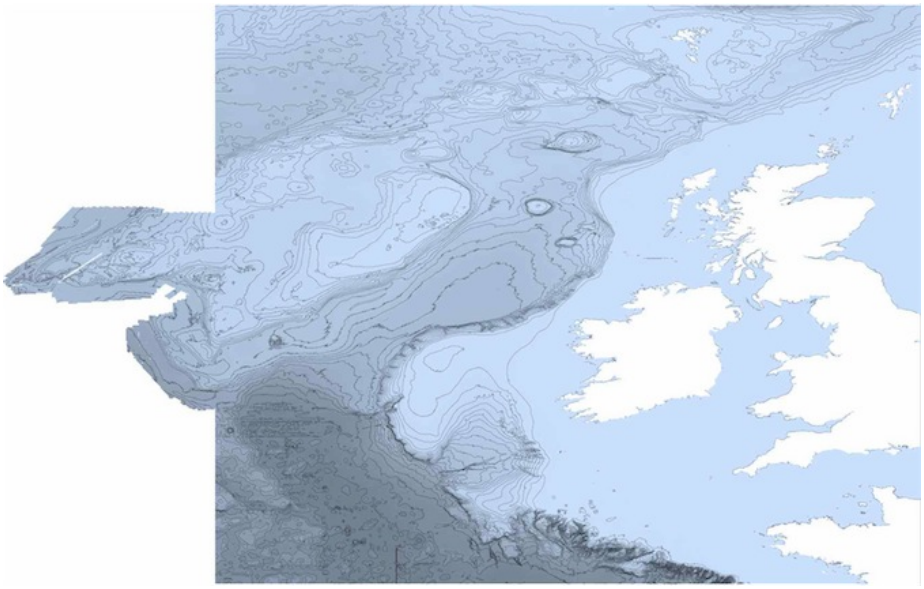
Here, we see that a table called `DEPTH` is available and that its dimensions are `[COLUMS,LINES]` (*i.e.* 6120 rows and 3840 columns). Extracting these data to create a `bathy` object is as simple as:

```
celt <- get.var.ncdf(nc, "DEPTH")
colnames(celt) <- get.var.ncdf(nc, "LINES")
rownames(celt) <- get.var.ncdf(nc, "COLUMNS")
class(celt) <- "bathy"
```

`celt` is now an object of class `bathy` on which we can use any `marmap` function, even if missing data are present. Here is the plot of this `bathy` object of more than 23.5 million cells:

```
# Custom color palette
blues <- colorRampPalette(c("lightsteelblue4", "lightsteelblue3",
                            "lightsteelblue2", "lightsteelblue1"))

# Map
plot(celt, image = TRUE, bpal = blues(100), lwd = 0.1)
```



This map appears slightly blurred here since we had to reduce the image quality to produce this vignette. The quality of the original map produced in `R` is much higher. Since the emodnet files have a resolution of 15 seconds, producing a map at such a large scale is time consumming. We thus recommend reading the "Working with big files" section of the `marmap-DataAnalysis` vignette for strategies to deal with such big datasets.

4 Getting bathymetric data from an xyz file

`read.bathy()` will read xyz data from any source. It can import bathymetric data for non rectangular areas or with lots of missing data as is often the case for custom datasets acquired by various types of sonar systems (*e.g.* Multibeam Echo Sounders). Alternatively, xyz files can be imported in `R` using `read.table()` and transformed to `bathy` objects with `as.bathy()`. Here, we will get ETOPO1 data [1] hosted on the NOAA GEODAS server [3]. 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 working directory (for example, with `setwd()`). Then laod the `marmap` package [4] with `library(marmap)` and 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)
papoue <- read.bathy('png.xyz', header = FALSE, sep = "\t")
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   -3120   -1540   -1640      -4    3710

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
```

5 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` [2] is installed and properly working.

```
library(RSQLite)

Loading required package:  DBI

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:

```
test <- subsetSQL(min_lon = 145, max_lon = 150,
                  min_lat = -2, max_lat = 0)
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   -3280   -2080   -2590   -1540      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, when you are done with the SQL dataset, you can remove it with:

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

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

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

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

[5] Pierce D (2014) ncdf: Interface to Unidata netCDF data files. URL http://CRAN.R-project.org/package=ncdf R package version 1.6.8.