Introduction to the 'chronosphere' R package

Adam T. Kocsis, Nussaïbah B. Raja

2021-04-16

1. Introduction

1.1. Installation

To install this alpha version of the package, you must download it either from the CRAN servers or its dedicated GitHub repository (https://github.com/chronosphere-portal/r_package/). All minor updates will be posted on GitHub as soon as they are finished, so please check this regularly. The version on CRAN will be lagging for some time, as it takes the servers many days to process everything. All questions and bugs can be reported at the GitHub issues board (https://github.com/chronosphere-portal/r_package/issues). Instead of spending it on actual research, a tremendous amount of time was invested in making this piece of software streamlined and user-friendly. If you use a dataset of the package in a publication, please cite both its reference(s) and the `chronosphere` package itself.

After installing from CRAN, from a source, or with `devtools::install_github()` , you can attach the package with:

```
library(chronosphere)
```

1.2 General features

The purpose of the `chronosphere` project is to facilitate, streamline and speed up the finding, acquisition and importing of Earth science data in R. Although the package currently focuses on deep time datasets, the scope of the included datasets will be much larger, spanning from a single variable published as supplementary material in a journal article to GIS data or the entire output of GCM models. `chronosphere` intends to decrease the gap between research hypotheses and the finding, downloading and importing of datasets.

2. Implemented classes

Faster data importing and better organization represents a considerable part of this process. Spatially explicit data are excellent candidates to demonstrate how more efficient data organization can speed up research. Although R has an excellent infrastructure for handling raster data (Hijmans & van Etten, 2019), the arrangement of individual layers are rather limited, which can be a problem when a large number of layers are considered. `RasterStack`s and `RasterBrick`s are very efficient for organizing `RasterLayer`s according to a single dimension (e.g. depth for 3D variables, or time), but more complicated structures frequently arise when more of these dimensions are neecessary. To offer a more effective solution, the `chronosphere` package includes the definition of the `RasterArray` and `SpatialArray` S4 classes.

`RasterArray`s represent hybrids between `RasterStack`s and regular R arrays. In short, they are virtual arrays of `RasterLayer`s, and can be thought of as regular arrays that include entire rasters as elements

rather than single `numeric`, `logical` or `character` values. As regular R users are familiar with subsetting, combinations and structures of regular arrays (including formal vectors and matrices), the finding, extraction and storage of spatially explicit data is much easier in such containers.

The `SpatialArray` class is analogous to the `RasterArray` only for vector data types implemented in the `sp` package (Pebesma and Bivand, 2005).

2.1. Structures

2.1.1. RasterArray

`RasterArray`s do not directly inherit from Raster* objects of the raster package, as a considerable number of main functions differ. They rather represent a wrapper object around regular RasterStacks - stacks of individual RasterLayers. This ensures that whenever users are unfamiliar with the methods of RasterArray class objects, they can always reduce their data to stacks or individual layers.

The formal class `RasterArray` has only two slots: `@stack` and `@index`. The stack includes the raster data in an unfolded manner, similarly to how matrices and arrays are essentially just vectors with additional attributes. The `@stack` slot incorporates a single `RasterStack` object, which represents the data content of the object. Accordingly, rasters in a `RasterArray` must have the same dimensions, extent and CRS. The `@index` slot, on the other hand, describes the structure of the `RasterArray`: it is a vector/matrix/array of integers each representing an index of the layers in the stack. The configuration (dimensions) of the index represents the entire array.

The `chronosphere` package includes one demo data sets: a set of ancient topographies (Scotese and Wright, 2018), which can be attached with the data command.

```
data(dems)
```

The structure of RasterArrays can be inspected if the object's name is typed into the console:

```
dems
```

```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 181, 361  (nrow, ncol)
## - resolution  : 1, 1  (x, y)
## - extent      : -180.5, 180.5, -90.5, 90.5  (xmin, xmax, ymin, ymax)
## - coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## Array properties:
## - dimensions   : 10  (vector)
## - num. layers    : 10
## - proxy:
##         0        5       10       15       20       25       30       35
##  "dem_0"  "dem_5" "dem_10" "dem_15" "dem_20" "dem_25" "dem_30" "dem_35"
##       40       45
## "dem_40" "dem_45"
```

The first part of the console output includes properties of the individual RasterLayers stored in the stack. These layers have to share essential attributes that allow them to be stored in a single stack (extent, resolution, CRS).

The second part of the output is a visualization of the structure of the RasterArray itself. In the case of the DEMs, 10 layers are stored in the stack, each layer having its individual name (e.g. `dem_0`). It is a single dimensional array (vector), and each element has its name in the array (`0`). Differentiating between the names of layers and the names of elements allows different subsetting and replacement rules for the two, which can both be handy - depending on the needs of the user.

The structure of the RasterArray can be visualized, analyzed or processed using the the proxy object. Proxies are essentially the same as the index slots of the RasterArray, but instead of including the indices of the layers they represent, proxies include the names of the layers. These can be accessed using the `proxy()` function.

```
proxy(dems)
```

```
##        0        5       10       15       20       25       30       35
##  "dem_0"  "dem_5" "dem_10" "dem_15" "dem_20" "dem_25" "dem_30" "dem_35"
##       40       45
## "dem_40" "dem_45"
```

Proxies are displayed as the second parts of the console output when the name of the object is typed into the console (show method).

RasterArrays are fairly easy to construct: you only need a stack of the data and a regular vector/matrix/array including integers. For instance, the dems object can be recreated from scratch without any problem.

```
# a stack of rasters
stackOfLayers <- dems@stack
# an index object
ind <- 1:10
names(ind) <- letters[1:10]
# a RasterArray
nra  <- RasterArray(index=ind, stack=stackOfLayers)
nra
```

```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 181, 361  (nrow, ncol)
## - resolution  : 1, 1  (x, y)
## - extent      : -180.5, 180.5, -90.5, 90.5  (xmin, xmax, ymin, ymax)
## - coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## Array properties:
## - dimensions   : 10  (vector)
## - num. layers    : 10
## - proxy:
##        a        b        c        d        e        f        g        h
##  "dem_0"  "dem_5" "dem_10" "dem_15" "dem_20" "dem_25" "dem_30" "dem_35"
##        i        j
## "dem_40" "dem_45"
```

The attributes of the index object are defining the structure of the `RasterArray`. `RasterArrays` can be created with the combination of individual `RasterLayers` (or `RasterArrays`) using the `combine()` function.

```
# one raster
r1 <- raster()
values(r1) <- 1
# same structure, different value
r2 <-raster()
values(r2) <- 2
comb <- combine(r1, r2)
comb
```

```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 180, 360  (nrow, ncol)
## - resolution  : 1, 1  (x, y)
## - extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
```

```
## - coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## Array properties:
## - dimensions   : 2  (vector)
## - num. layers    : 2
## - proxy:
##        r1        r2
## "layer.1" "layer.2"
```

Matrix-like RasterArrays can also be created easily with the `cbind()` and `rbind()` functions.

```
# bind dems to itself
cbind(dems, dems)
```

```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 181, 361  (nrow, ncol)
## - resolution  : 1, 1  (x, y)
## - extent      : -180.5, 180.5, -90.5, 90.5  (xmin, xmax, ymin, ymax)
## - coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## Array properties:
## - dimensions  : 10, 2  (nrow, ncol)
## - num. layers    : 20
## - proxy:
##     [,1]       [,2]
## 0  "dem_0.1"  "dem_0.2"
## 5  "dem_5.1"  "dem_5.2"
## 10 "dem_10.1" "dem_10.2"
## 15 "dem_15.1" "dem_15.2"
## 20 "dem_20.1" "dem_20.2"
## 25 "dem_25.1" "dem_25.2"
## 30 "dem_30.1" "dem_30.2"
## 35 "dem_35.1" "dem_35.2"
## 40 "dem_40.1" "dem_40.2"
## 45 "dem_45.1" "dem_45.2"
```

2.1.2. SpatialArray

The `SpatialArray`s also do not inherit from the respective `Spatial*` classes. This class has the same structure as the `RasterArray` containing an `@index` and a `@stack` slot. The difference between `SpatialArray`s and `RasterArray`s is that the `@slack` slot includes a class named `SpatialStack`, which is a formalized `list` of `Spatial*` objects, and has a similar interface to a `RasterStack`. `SpatialStack`s can include any of the `SpatialPoints`,`SpatialPointsDataFrame`,`SpatialLines`,`SpatialLinesDataFrame`,`SpatialPolygons` or `SpatialPolygonsDataFrame` classes, as long as they have the same CRS.

The package includes one demo dataset, the first five entries of the PaleoMAP Paleocoastlines (Kocsis and Scotese, 2020).

```
data(coasts)
coasts
```

```
## class          : SpatialArray
## Spatial* properties:
## - types        : SpatialPolygonsDataFrame
## - bbox         : -180, 180, -90, 89.95514  (xmin, xmax, ymin, ymax)
## - coord. ref.  :  +proj=longlat +ellps=WGS84
## Array properties:
```

```
## - dimensions  : 5, 2  (nrow, ncol)
## - num. layers  : 10
## - proxy:
##     margin        coast
## 0  "X0Ma_CM_v7"  "X0Ma_CS_v7"
## 5  "X5Ma_CM_v7"  "X5Ma_CS_v7"
## 10 "X10Ma_CM_v7" "X10Ma_CS_v7"
## 15 "X15Ma_CM_v7" "X15Ma_CS_v7"
## 20 "X20Ma_CM_v7" "X20Ma_CS_v7"
```

This is a two dimensional SpatialArray that has five rows (ages) and two columns: the outlines of the continental margins (`margins`) and the coastlines (`coast`). With the `proxy()` function it is easy to interact with this object, or to query or analyze it.

```r
proxy(coasts)
```

```
##    margin        coast
## 0  "X0Ma_CM_v7"  "X0Ma_CS_v7"
## 5  "X5Ma_CM_v7"  "X5Ma_CS_v7"
## 10 "X10Ma_CM_v7" "X10Ma_CS_v7"
## 15 "X15Ma_CM_v7" "X15Ma_CS_v7"
## 20 "X20Ma_CM_v7" "X20Ma_CS_v7"
```

Creating a `SpatialArray` from scratch is as easy as that of a `RasterArray` using the `combine` function that can be applied to a number of `Spatial*` objects.

```r
# Individual SP objects
margin0 <- coasts[1, 1]
margin5 <- coasts[2, 1]
combination<- combine(margin0, margin5)
combination
```

```
## class          : SpatialArray
## Spatial* properties:
## - types        : SpatialPolygonsDataFrame
## - bbox         : -179.999, 179.9998, -89.99999, 89.95514  (xmin, xmax, ymin, ymax)
## - coord. ref.  :  +proj=longlat +ellps=WGS84
## Array properties:
## - dimensions   : 2  (vector)
## - num. layers  : 2
## - proxy:
##  margin0 margin5
##    "X1"    "X2"
```

Alternatively, they can be rebuilt using lower level manipulation, by first creating a `SpatialStack`

```r
coast0 <- coasts[1, 2]
coast5 <- coasts[1, 2]
# create SpatialStack
spStack<- stack(margin0, margin5, coast0, coast5)
# alternatively: accepts filenames too
# spStack <- SpatialStack(list(margin0, margin5, coast0, coast5))
```

and then fold it appropriately with an index

```r
ind <- matrix(1:4, ncol=2)
colnames(ind) <- c("margin", "coast")
rownames(ind) <- c("0", "5")
```

```
spArray <- SpatialArray(index=ind, stack=spStack)
spArray
```

```
## class          : SpatialArray
## Spatial* properties:
## - types        : SpatialPolygonsDataFrame
## - bbox         : -180, 180, -90, 89.95514  (xmin, xmax, ymin, ymax)
## - coord. ref.  :  +proj=longlat +ellps=WGS84
## Array properties:
## - dimensions  : 2, 2  (nrow, ncol)
## - num. layers  : 4
## - proxy:
##    margin coast
## 0 "X1"   "X3"
## 5 "X2"   "X4"
```

2.2. RasterArray and SpatialArray attributes and functions to query

Functions that query and change attributes of the `RasterArray` resemble general arrays more than Raster* objects. They are connected to the `@index` slots, which are shared by the `RasterArray` and `SpatialArray` objects. Therefore, the dimensional organization of the two classes, and accordingly, some methods are shared (written for the `XArray` class-union).

The number of elements represented in the RasterArray can be queried with the `length()` function:

```
length(dems)
```

```
## [1] 10
```

This RasterArray has 10 elements. The number of column and row names can be queried in a similar way:

```
nrow(coasts)
```

```
## [1] 5
```

```
ncol(coasts)
```

```
## [1] 2
```

These functions are summarized in the dim() function. This, however, unlike the regular `dim()` method of vectors, return the length of the RasterArray vector, rather than just `NULL`.

```
dim(dems)
```

```
## [1] 10
```

```
dim(coasts)
```

```
## [1] 5 2
```

The organization of RasterLayers can be greatly facilitated with names. The `names()`, `colnames()`, `rownames()` and `dimnames()` functions work the same way on RasterArrays as if they were arrays of simple numeric, logical or character values. The `names()` function returns the names of individual elements of a vector-like RasterArray.

```
names(dems)
```

```
##  [1] "0"  "5"  "10" "15" "20" "25" "30" "35" "40" "45"
```

The colnames() and rownames() functions are more relevant for matrix-like RasterArrays or SpatialArrays, such as coast.

```
colnames(coasts)
```

```
## [1] "margin" "coast"
```

```
rownames(coasts)
```

```
## [1] "0"  "5"  "10" "15" "20"
```

All name-related methods can be used for replacement as well. For instance, you can quickly rename the names of the columns of the coast object this way:

```
coasts2 <- coasts
colnames(coasts2) <- c("m", "c")
coasts2
```

```
## class          : SpatialArray
## Spatial* properties:
## - types        : SpatialPolygonsDataFrame
## - bbox         : -180, 180, -90, 89.95514  (xmin, xmax, ymin, ymax)
## - coord. ref.  :  +proj=longlat +ellps=WGS84
## Array properties:
## - dimensions  : 5, 2  (nrow, ncol)
## - num. layers  : 10
## - proxy:
##     m             c
## 0  "X0Ma_CM_v7"  "X0Ma_CS_v7"
## 5  "X5Ma_CM_v7"  "X5Ma_CS_v7"
## 10 "X10Ma_CM_v7" "X10Ma_CS_v7"
## 15 "X15Ma_CM_v7" "X15Ma_CS_v7"
## 20 "X20Ma_CM_v7" "X20Ma_CS_v7"
```

Just as you would do with normal arrays, you can query/rewrite all names with the dimnames() function, which uses a list to store the names in every dimension.

```
dimnames(coasts2)[[1]] <- paste(rownames(coasts), "Ma")
coasts2
```

```
## class          : SpatialArray
## Spatial* properties:
## - types        : SpatialPolygonsDataFrame
## - bbox         : -180, 180, -90, 89.95514  (xmin, xmax, ymin, ymax)
## - coord. ref.  :  +proj=longlat +ellps=WGS84
## Array properties:
## - dimensions  : 5, 2  (nrow, ncol)
## - num. layers  : 10
## - proxy:
##        m             c
## 0 Ma  "X0Ma_CM_v7"  "X0Ma_CS_v7"
## 5 Ma  "X5Ma_CM_v7"  "X5Ma_CS_v7"
## 10 Ma "X10Ma_CM_v7" "X10Ma_CS_v7"
## 15 Ma "X15Ma_CM_v7" "X15Ma_CS_v7"
## 20 Ma "X20Ma_CM_v7" "X20Ma_CS_v7"
```

Besides the names of the elements in the RasterArray or SpatialArray, every layer has its own name in the stack. These can be accessed with layers() function:

```r
layers(dems)
```

```
##  [1] "dem_0"  "dem_5"  "dem_10" "dem_15" "dem_20" "dem_25" "dem_30" "dem_35"
##  [9] "dem_40" "dem_45"
```

The total number of cells in the `RasterLayer` or the entire stack can accessed with the `ncell()` and `nvalues()` functions, respectively.

```r
ncell(dems)
```

```
## [1] 65341
```

```r
nvalues(dems)
```

```
## [1] 653410
```

2.3. Subsetting and replacement

Facilitating the accession of items is the primary purpose of `RasterArrays` and `SpatialArrays`. These either focus on the layers (stack items, double bracket operator "[[") or on the elements of the RasterArray (single bracket operator "[").

2.3.1 Layer selection - Double bracket [[

This form of subsetting and replacement is inherited from the RasterStack class. Individual layers can be accessed directly from the stack using either the position index, the name of the layer, or the logical value pointing to the position. Whichever is used, the RasterArray wrapper is omitted and the output will be a RasterLayer or RasterStack class object.

A single layer can be accessed using its name, regardless of its position in the RasterArray. This can be visualized either with the default `plot()` or the more general `mapplot()` function.

```r
one <- dems[["dem_45"]]
mapplot(one, col="earth")
```

The double bracket returns a single RasterArray. Using multiple elements for subsetting will return a RasterStack:

```
dems[[c(1,2)]]
```

```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 181, 361  (nrow, ncol)
## - resolution  : 1, 1  (x, y)
## - extent      : -180.5, 180.5, -90.5, 90.5  (xmin, xmax, ymin, ymax)
## - coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## Array properties:
## - dimensions   : 2  (vector)
## - num. layers    : 2
## - proxy:
##  [1] "dem_0" "dem_5"
```

These are the first two RasterLayers in the stack of the RasterArray.

Double brackets can also be used for replacements, but as this has no effect on the structure of the array, changes implemented with this method are more difficult to trace. For instance,

```r
# copy
dem2 <- dems
dem2[["dem_0"]] <- dem2[["dem_5"]]
```

will rewrite the values in the first element of `dem2`, but that will not be evident in the RasterArray's structure.

```r
# but these two are now the same
dem2[[1]]
```

```
## class      : RasterLayer
## dimensions : 181, 361, 65341  (nrow, ncol, ncell)
## resolution : 1, 1  (x, y)
## extent     : -180.5, 180.5, -90.5, 90.5  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## source     : memory
## names      : dem_0
## values     : -7000, 6300  (min, max)
## zvar       : z
```

```r
dem2[[2]]
```

```
## class      : RasterLayer
## dimensions : 181, 361, 65341  (nrow, ncol, ncell)
## resolution : 1, 1  (x, y)
## extent     : -180.5, 180.5, -90.5, 90.5  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## source     : memory
## names      : dem_5
## values     : -7000, 6300  (min, max)
## zvar       : z
```

2.3.2 Single bracket

Features offered by the double bracket ("[[") operator are virtually identical with those of RasterStacks. The true utility of RasterArrays become evident with simple array-type subsetting.

Unlike Raster* objects of the raster package, single brackets will retrieve and replace items from the RasterArray as if they were simple arrays. For example, single elements of the DEMs can be selected with the age of the DEM, passed as a character subscript.

```r
dems["30"]
```

```
## class      : RasterLayer
## dimensions : 181, 361, 65341  (nrow, ncol, ncell)
## resolution : 1, 1  (x, y)
## extent     : -180.5, 180.5, -90.5, 90.5  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## source     : memory
## names      : dem_30
## values     : -8000, 10200  (min, max)
## zvar       : z
```

returning the 30Ma RasterLayer. By default, the RasterArray container is dropped, but it can be conserved if the drop argument is set to `FALSE`.

```r
dem30 <- dems["30", drop=FALSE]
class(dem30)
```

```
## [1] "RasterArray"
## attr(,"package")
## [1] "chronosphere"
```

Accessing elements which are not present is valid for single dimensional RasterArrays (vector-like ones):

```
dems[4:12]
```

```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 181, 361  (nrow, ncol)
## - resolution  : 1, 1  (x, y)
## - extent      : -180.5, 180.5, -90.5, 90.5  (xmin, xmax, ymin, ymax)
## - coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## Array properties:
## - dimensions   : 9  (vector)
## - num. layers    : 7
## - proxy:
##        15       20       25       30       35       40       45     <NA>
## "dem_15" "dem_20" "dem_25" "dem_30" "dem_35" "dem_40" "dem_45"       NA
##     <NA>
##       NA
```

Missing values are legitimate parts of RasterArrays. These gaps in the data are not represented in the stacks but only in the index slots of the RasterArrays. They can be inserted or added into the layers.

```
demna <- dems
demna[3] <- NA
```

Multidimensional subscripts work in a similar fashion. If a single layer is desired from the `RasterArray` or `SpatialArray`, that can be accessed using the names of the margins.

```
# character type is necessary as the rowname is "2003"
one <- coasts["15", "coast"]
mapplot(one)
```

similarly to entire rows,

```
coasts["15", ]
```

```
## class          : SpatialArray
## Spatial* properties:
## - types        : SpatialPolygonsDataFrame
## - bbox         : -180, 180, -90, 88.43112  (xmin, xmax, ymin, ymax)
## - coord. ref.  :  +proj=longlat +ellps=WGS84
## Array properties:
## - dimensions   : 2  (vector)
## - num. layers  : 2
## - proxy:
##         margin         coast
## "X15Ma_CM_v7" "X15Ma_CS_v7"
```

or columns

```
coasts[,"coast"]
```

```
## class          : SpatialArray
## Spatial* properties:
```

```
## - types        : SpatialPolygonsDataFrame
## - bbox         : -180, 180, -90, 83.77569  (xmin, xmax, ymin, ymax)
## - coord. ref.  :  +proj=longlat +ellps=WGS84
## Array properties:
## - dimensions   : 5  (vector)
## - num. layers  : 5
## - proxy:
##              0             5            10            15            20
##  "X0Ma_CS_v7"  "X5Ma_CS_v7" "X10Ma_CS_v7" "X15Ma_CS_v7" "X20Ma_CS_v7"
```

2.4 Functionality inherited from raster and sp

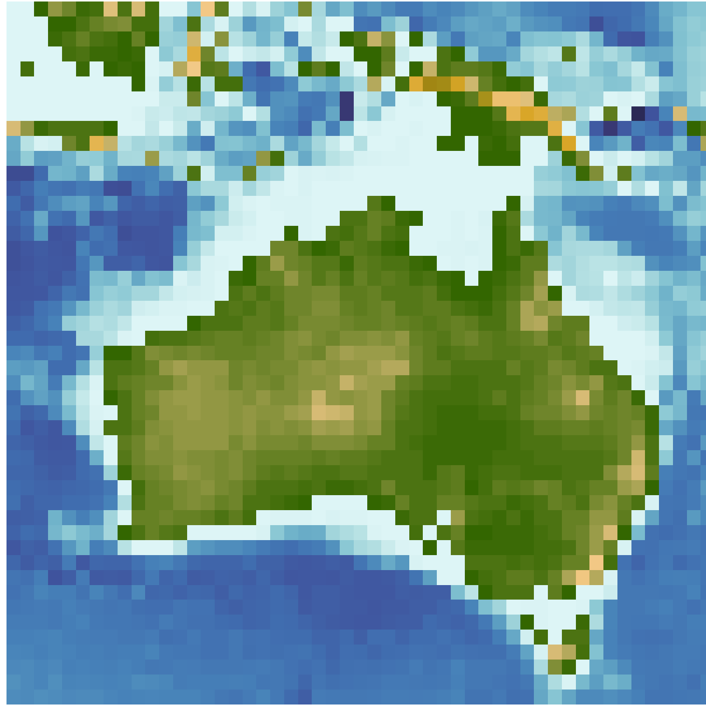
2.4.1. `RasterArray`

As the spatial information is contained entirely in the `RasterStack`s, a number of methods are practically inherited from `RasterStack` class. For instance, all `RasterLayer`s of the `RasterArray` can be cropped in a single line of code.

```
# crop to Australia
ext <- extent(c(
  xmin = 106.58,
  xmax = 157.82,
  ymin = -45.23,
  ymax = 1.14
))

# cropping all DEMS
au<- crop(dems, ext)

# select the first element
mapplot(au[1], col="earth")
```



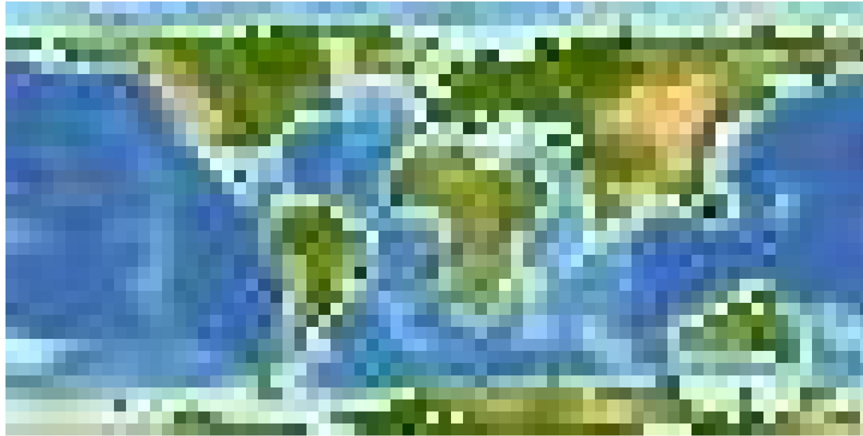
If you plot all cropped DEMs, you can see how Australia drifted to its current position.

Other functions such as aggregation or resampling work in a similar way.

```
template <- raster(res=5)

# resample all DEMS
coarse <- resample(dems, template)

# plot an element
mapplot(coarse["45"], col="earth")
```



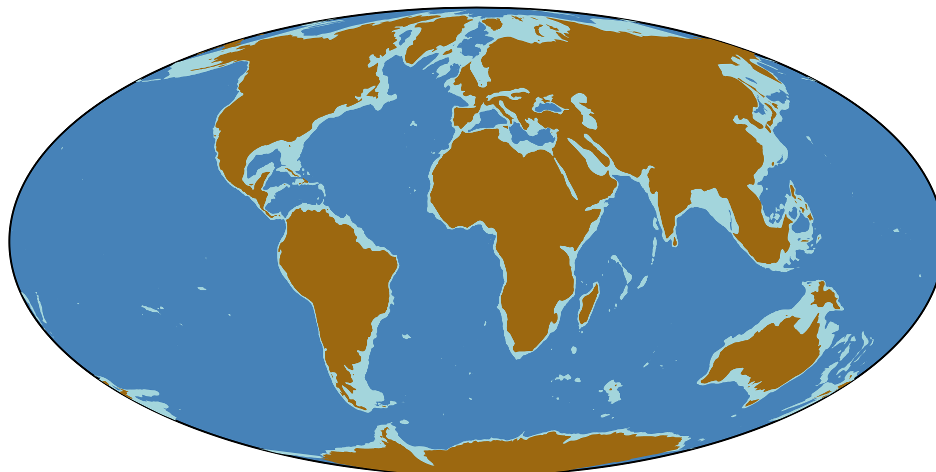
2.4.2. SpatialArray

Similar rules apply to SpatialArrays as to RasterArrays. For example, projection changes can be implemented for the entire set, if the rgdal packgae is installed on the system:

```
# Cylindrial equal area projection
mollCoasts <- spTransform(coasts, CRS("+proj=moll"))

# plot edge of the map
edge<- spTransform(mapedge(), CRS("+proj=moll"))
plot(edge, col=ocean(8)[5])

# the entire thing
plot(mollCoasts["20", "margin"], col=ocean(8)[7], border=NA, add=T)
plot(mollCoasts["20", "coast"], col=terra(8)[5], add=T, border=NA)
```



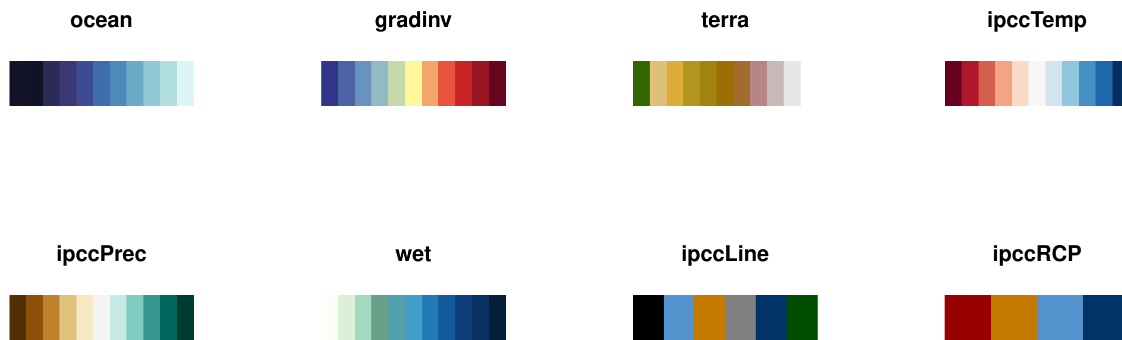
3. Plotting

The `mapplot()` function makes it easy to produce visually pleasing plots of a Raster* object.

3.1 Colour palettes

The package includes several colour palettes which can be used for plotting purposes. An additional palette option developed specifically with DEMs in mind is *earth*. This combines the *ocean* and *terra* palettes and automatically sets the breaks to differentiate between marine and terrestrial cells. An example is shown below in RasterLayer plotting example.

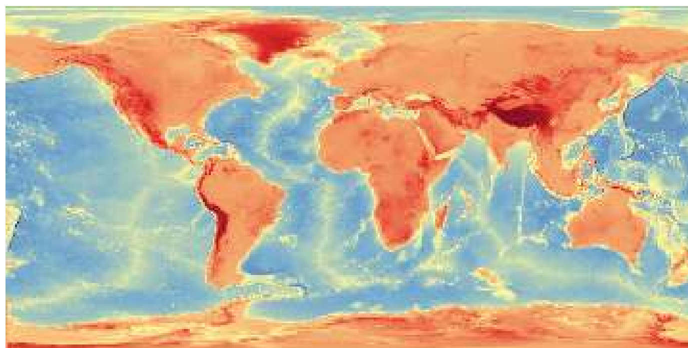
```
showPal()
```



3.2 RasterLayer

The `mapplot()` function for RasterLayer works similar to the `plot()` function. The `mapplot` function for the Raster* objects include a default palette and omits the legend, axes and bounding box.

```
data(dems)
mapplot(dems[1])
```



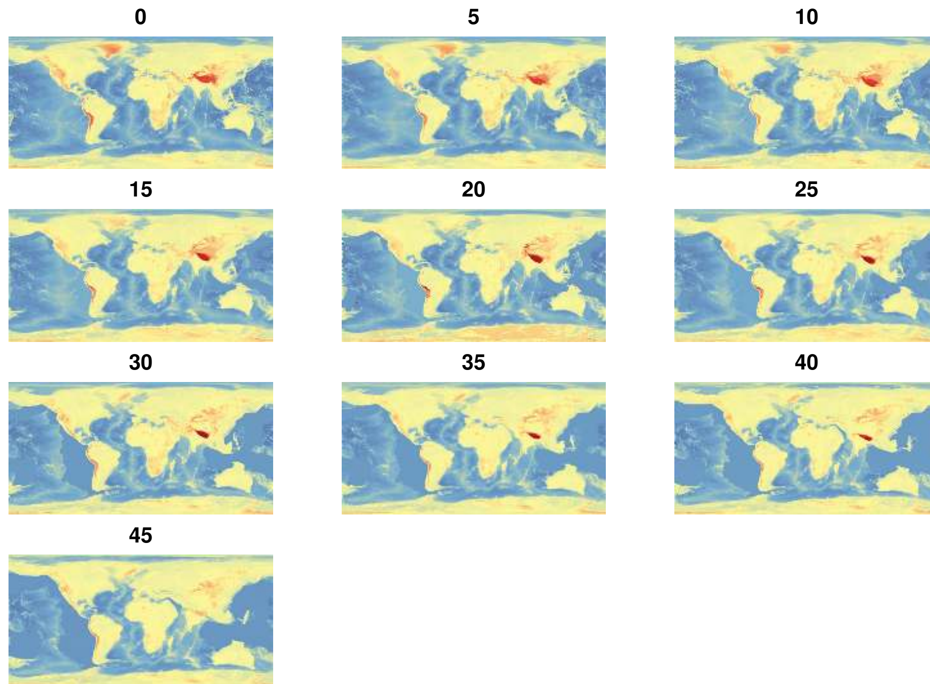
```
#using a custom colour palette
mapplot(dems[1], col="earth", main="Using the earth palette")
```

Using the earth palette

3.3 RasterArray

RasterArrays can be plotted as multi-faceted plots using `mapplot()`. The `mapplot()` function keeps the structure of the RasterArray in terms of the order that the plots are generated. The plot titles are automatically generated based on the names of the layers within the RasterArray. This can be changed by passing the custom plot titles to the argument `plot.title`. **Note:** The number of plot titles provided should be equal to the number of layers in the RasterArray.

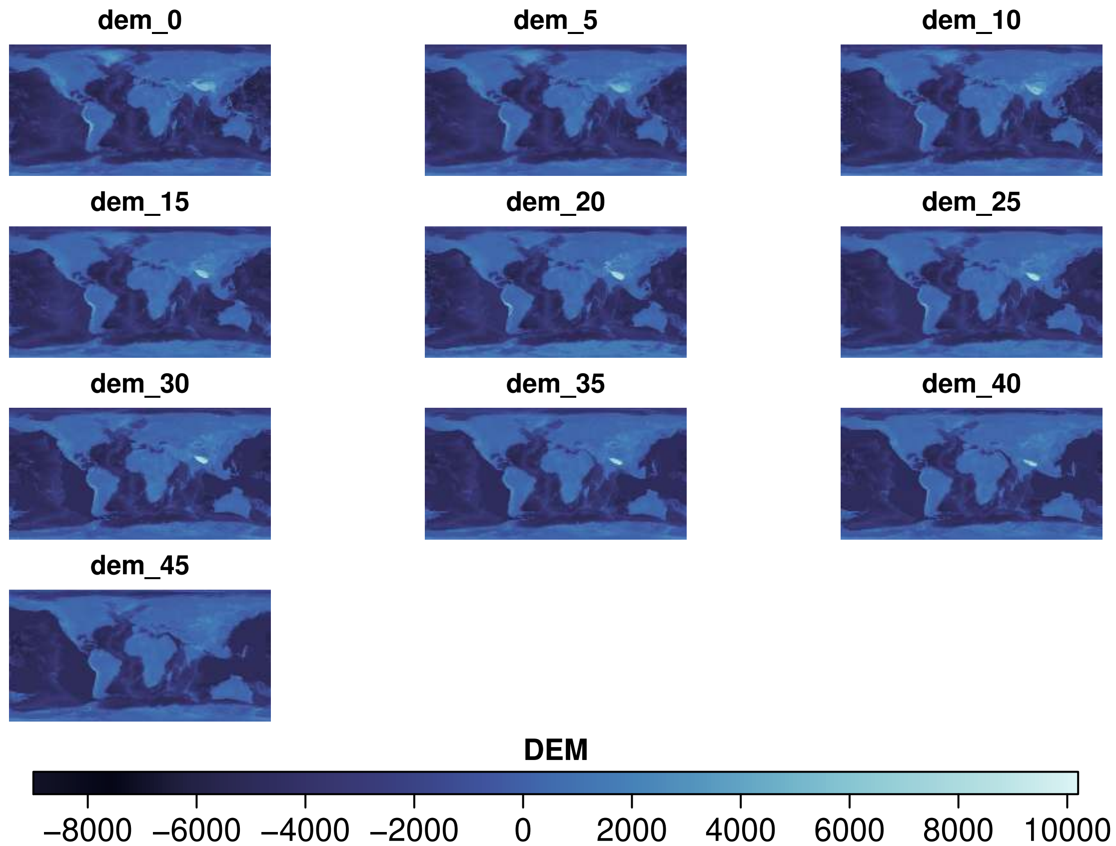
```
data(dems)
mapplot(dems)
```



3.3.1 Adding a single consistent legend

Just like with RasterLayers, different palettes can be used for the plots. This implements a single palette for all the plots. In addition, this can then be used to generate a single legend that ensures consistency and makes it easier to compare the figures.

```
data(dems)
mapplot(dems, col="ocean", legend=TRUE, legend.title="DEM",
        plot.title=proxy(dems))
```



3.3.2 RasterArrays with NA layers

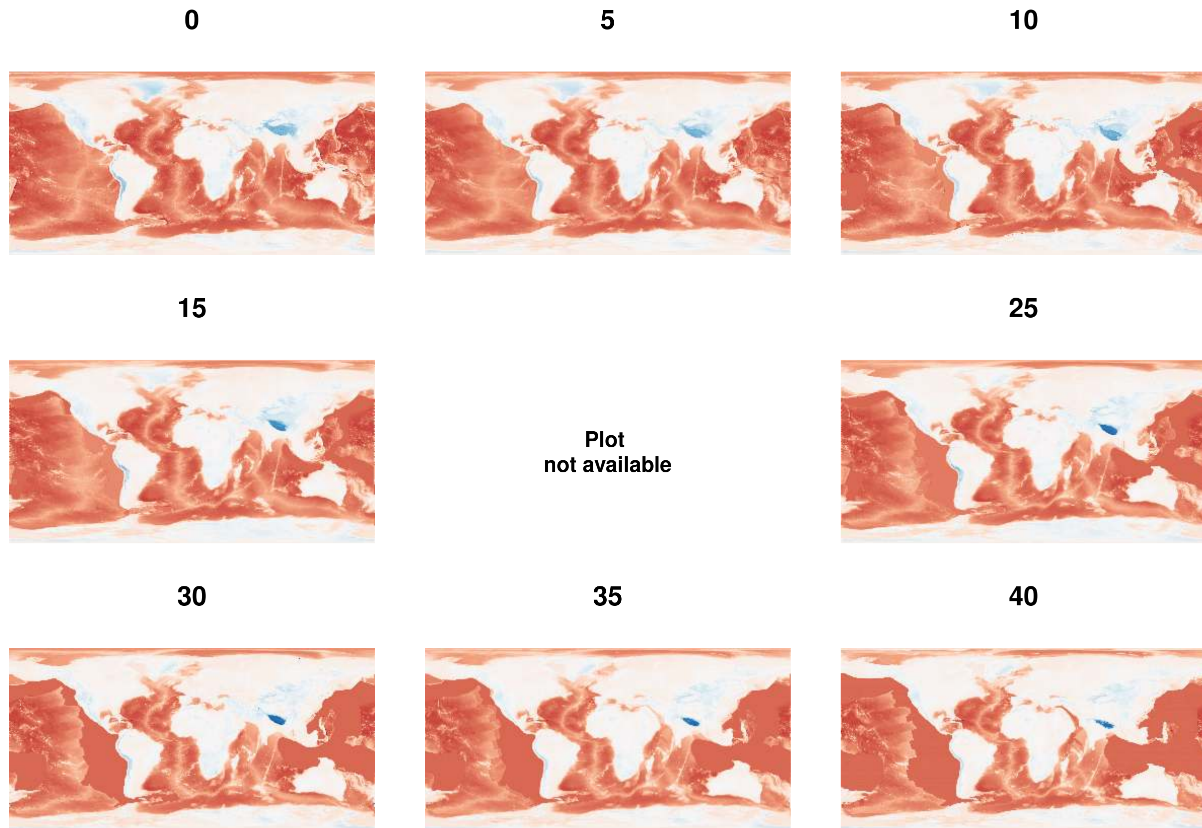
In the event that one of the layers of the dem does not exist, i.e. has an NA value instead, the plot corresponding to this NA layer will show "Plot not available".

```
data(dems)

#create NA layer
dems[5] <- NA
is.na(dems)
```

```
##     0     5    10    15    20    25    30    35    40    45
## FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

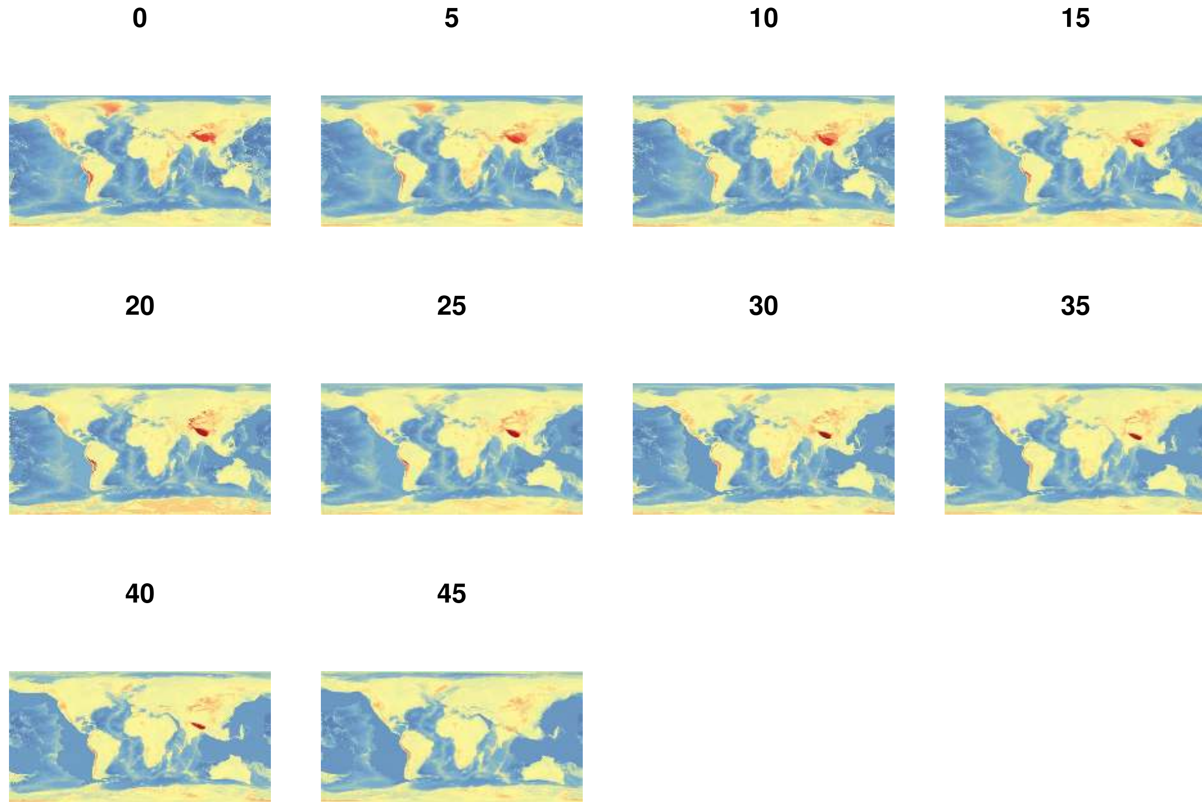
```
mapplot(dems, col="ipccTemp")
```



3.3.3 Number of columns and pages

The custom number of columns can also be specified through the `ncol` argument. The default number of columns is 3.

```
# 4 columns
data(dems)
mapplot(dems, ncol=4)
```



Sometimes, there might be too many plots to plot all on one single page. In this instance, the argument `multi` can be set to `TRUE` to allow the maps to be plotted in multiple figures.

```
data(dems)
mapplot(dems, multi=TRUE)
```

References

Hijmans, R. J., & van Etten, J. (2019). raster: Geographic Data Analysis and Modeling. Retrieved from https://cran.r-project.org/package=raster

Pebesma, E. J., & Bivand, R. S. (2005). Classes and methods for spatial data in R. R News, 5(2), 9–13.

Kocsis, A. T., & Scotese, C. R. (2020). PaleoMAP PaleoCoastlines data [Data set]. Zenodo. https://doi.org/10.5281/zenodo.3903164

Scotese, C. R. Wright, N. (2018). PALEOMAP Paleodigital Elevation Models (PaleoDEMS) for the Phanerozoic. URL: https://www.earthbyte.org/paleodem-resource-scotese-and-wright-2018/