Introduction to the 'chronosphere' R package

Adam T. Kocsis, Nussaïbah B. Raja

2019-11-29

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 (http://www.github.com/adamkocsis/chronosphere/). 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/adamkocsis/chronosphere/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 the 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 fasten the finding, acquisition and importing of Earth science data in R. Although the package currently focuses on deep time data sets, the scope of the included data sets 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, download and importing of data sets.

2. RasterArray

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 have to be considered. RasterStacks and RasterBricks are very efficient for organizing RasterLayers according to a single dimension (e.g. depth for 3D variables, or time), multidimensional structures are preferred.

To offer a more effective solution, the `chronosphere` package includes the definition of the RasterArray S4 class. RasterArrays represent hybrids between RasterStacks and regular R arrays. In short, they are virtual arrays of RasterLayers, 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.

2.1. Structure

RasterArrays do not directly inherit from Raster* objects of the raster package, as a considerable number of main functions differ, but 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: a stack and 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. 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 two demo data sets: a set of ancient topographies (Scotese and Wright, 2018) and time series of bio-climatic variables (annual mean temperature and precipitation) from the CHELSA project (Karger et al, 2017a, 2017b) These datasets can be attached with the data command.

```
data(dems)
data(clim)
```

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`). The differentiating between the names of layers and the names of elements allows different subsetting and replacement rules for the two, which both can 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).

```
clim
```

```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 90, 180  (nrow, ncol)
## - resolution  : 2, 2  (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  : 10, 2  (nrow, ncol)
## - num. layers    : 20
## - proxy:
##       bio1         bio12
## 2001 "bio01_2001" "bio12_2001"
## 2002 "bio01_2002" "bio12_2002"
## 2003 "bio01_2003" "bio12_2003"
## 2004 "bio01_2004" "bio12_2004"
## 2005 "bio01_2005" "bio12_2005"
## 2006 "bio01_2006" "bio12_2006"
## 2007 "bio01_2007" "bio12_2007"
## 2008 "bio01_2008" "bio12_2008"
## 2009 "bio01_2009" "bio12_2009"
## 2010 "bio01_2010" "bio12_2010"
```

This RasterArray has 10 rows (annual means) and two variables/columns: temperature (`bio1`) and precipitation (`bio12`). With the `proxy()` function it is easy to interact with this object, or to query or analyze it.

```
proxy(clim)
```

```
##      bio1         bio12
## 2001 "bio01_2001" "bio12_2001"
## 2002 "bio01_2002" "bio12_2002"
## 2003 "bio01_2003" "bio12_2003"
## 2004 "bio01_2004" "bio12_2004"
## 2005 "bio01_2005" "bio12_2005"
## 2006 "bio01_2006" "bio12_2006"
## 2007 "bio01_2007" "bio12_2007"
## 2008 "bio01_2008" "bio12_2008"
## 2009 "bio01_2009" "bio12_2009"
## 2010 "bio01_2010" "bio12_2010"
```

RasterArrays are fairly easy to construct: one only needs a stack of the data and an 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"
```

2.2. RasterArray attributes and function to query

Functions that query and change attributes of the RasterArray resemble general arrays more than Raster* objects. They are connected to the index slot of the RasterArray and return values accordingly.

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(clim)
```

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

```
ncol(clim)
```

```
## [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(clim)
```

```
## [1] 10  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, such as `clim`.

```
colnames(clim)
```

```
## [1] "bio1"  "bio12"
```

```
rownames(clim)
```

```
##  [1] "2001" "2002" "2003" "2004" "2005" "2006" "2007" "2008" "2009" "2010"
```

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

```
clim2 <- clim
colnames(clim2) <- c("temp", "prec")
clim2
```

```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 90, 180  (nrow, ncol)
## - resolution  : 2, 2  (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  : 10, 2  (nrow, ncol)
## - num. layers    : 20
## - proxy:
##       temp         prec
## 2001 "bio01_2001" "bio12_2001"
## 2002 "bio01_2002" "bio12_2002"
## 2003 "bio01_2003" "bio12_2003"
## 2004 "bio01_2004" "bio12_2004"
## 2005 "bio01_2005" "bio12_2005"
## 2006 "bio01_2006" "bio12_2006"
## 2007 "bio01_2007" "bio12_2007"
## 2008 "bio01_2008" "bio12_2008"
## 2009 "bio01_2009" "bio12_2009"
## 2010 "bio01_2010" "bio12_2010"
```

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

```
dimnames(clim2)[[1]] <- 1:10
clim2
```

```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 90, 180  (nrow, ncol)
## - resolution  : 2, 2  (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  : 10, 2  (nrow, ncol)
## - num. layers    : 20
## - proxy:
##     temp         prec
## 1  "bio01_2001" "bio12_2001"
## 2  "bio01_2002" "bio12_2002"
## 3  "bio01_2003" "bio12_2003"
## 4  "bio01_2004" "bio12_2004"
## 5  "bio01_2005" "bio12_2005"
## 6  "bio01_2006" "bio12_2006"
## 7  "bio01_2007" "bio12_2007"
## 8  "bio01_2008" "bio12_2008"
## 9  "bio01_2009" "bio12_2009"
## 10 "bio01_2010" "bio12_2010"
```

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

```
layers(clim)
```

```
##  [1] "bio01_2001" "bio01_2002" "bio01_2003" "bio01_2004" "bio01_2005"
##  [6] "bio01_2006" "bio01_2007" "bio01_2008" "bio01_2009" "bio01_2010"
## [11] "bio12_2001" "bio12_2002" "bio12_2003" "bio12_2004" "bio12_2005"
## [16] "bio12_2006" "bio12_2007" "bio12_2008" "bio12_2009" "bio12_2010"
```

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

```
ncell(dems)
```

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

```
nvalues(dems)
```

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

2.3. Subsetting and replacement

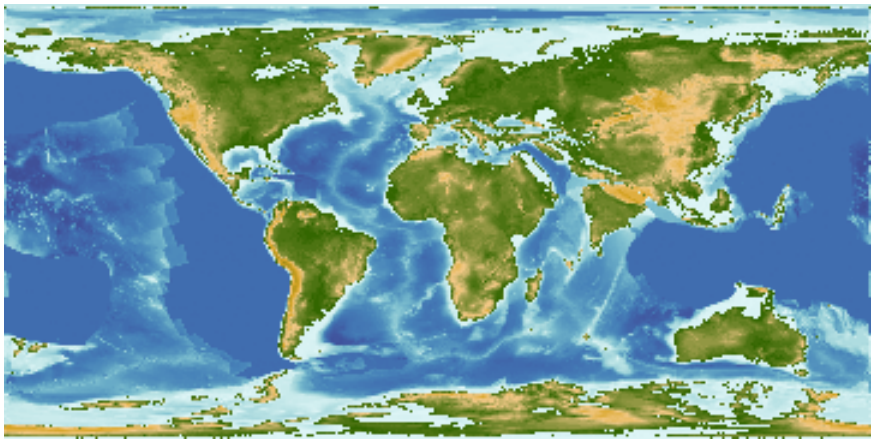
Facilitating the accession of items is the primary purpose of RasterArrays. These either focus on the layers (stack items, double bracket operator "[[") or the elements of the RasterArray (single bracket operator "[").

2.3.1 Layer selection - Double bracket [[

This form of subsetting and replacement are 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 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.

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



Returning a single RasterArray. Multiple elements will be the format of a stack:

```
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"
```

Which 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,

```
# 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.

```
# 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
```

```
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 get 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.

```
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`.

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

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

Beyond bounds accessing is valid for single dimensional RasterArrays (vector-like ones):

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

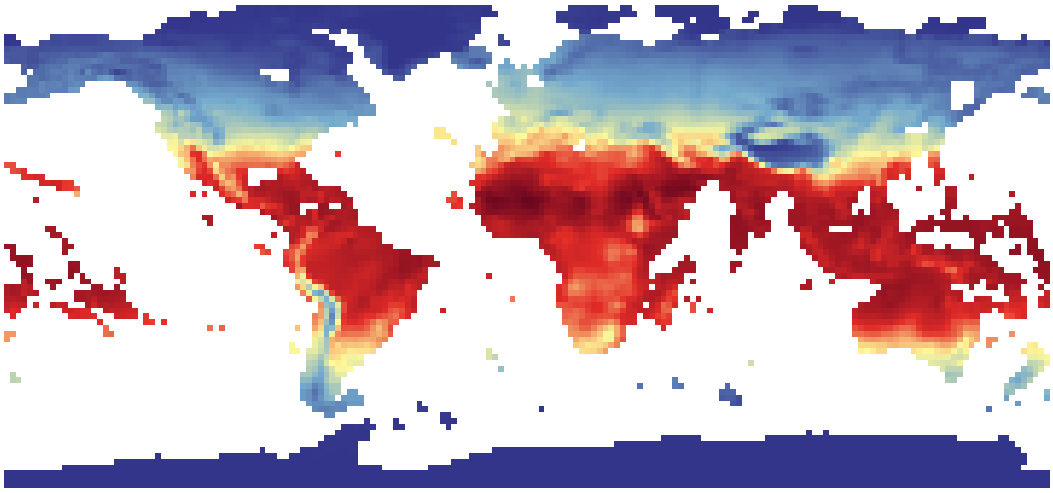
```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 181, 361  (nrow, ncol)
## - resolution  : 1, 1  (x, y)
## - extent      : -180.5, 180.5, -90.5, 90.5  (xmin, xmax, ymin, ymax)
## - coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## Array properties:
## - 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, that can be accessed using the names of the margins.

```
# character is necessary, as the row named "2003" is necessary
one <- clim["2003", "bio1"]
mapplot(one)
```



similarly to entire rows,

```
clim["2005", ]
```

```
## class         : RasterArray
## RasterLayer properties:
## - dimensions  : 90, 180  (nrow, ncol)
## - resolution  : 2, 2  (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:
##          bio1        bio12
## "bio01_2005" "bio12_2005"
```

or columns

```
clim[,"bio12"]
```

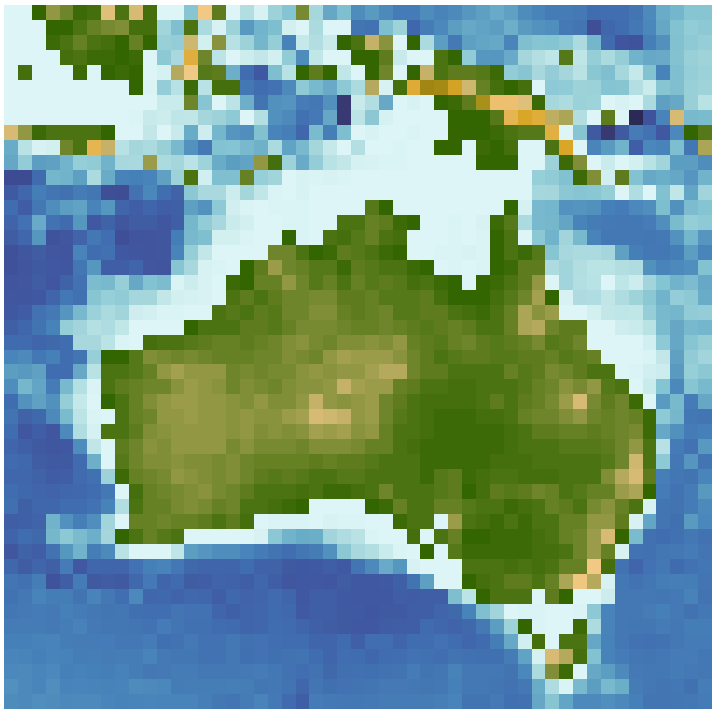
```
## class         : RasterArray
```

```
## RasterLayer properties:
## - dimensions  : 90, 180  (nrow, ncol)
## - resolution  : 2, 2  (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   : 10  (vector)
## - num. layers    : 10
## - proxy:
##          2001         2002         2003         2004         2005         2006
## "bio12_2001" "bio12_2002" "bio12_2003" "bio12_2004" "bio12_2005" "bio12_2006"
##         2007         2008         2009         2010
## "bio12_2007" "bio12_2008" "bio12_2009" "bio12_2010"
```

2.4 Inherited from Raster*

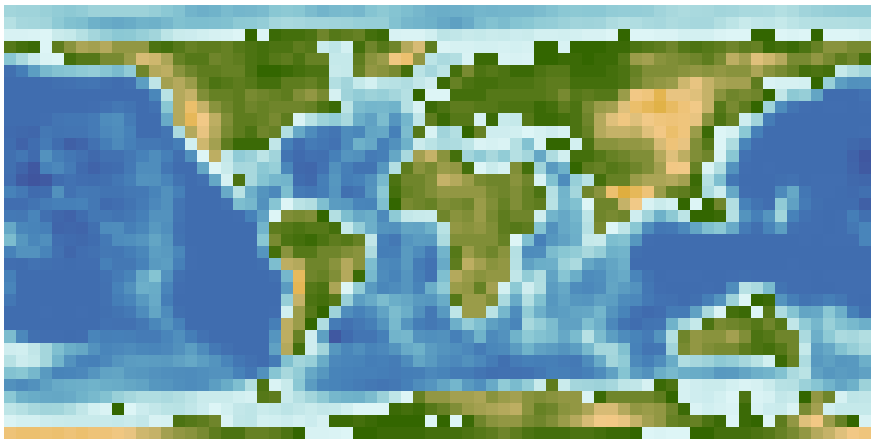
As the spatial information is contained entirely in the RasterStacks, a number methods are practically inherited from RasterStack class. For instance, all RasterLayer of the RasterArray can be cropped in a single line of code.

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

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

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



Other functions such as aggregation or resampling works just the same.

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

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

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



3. Plotting

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

3.1 Colour palettes

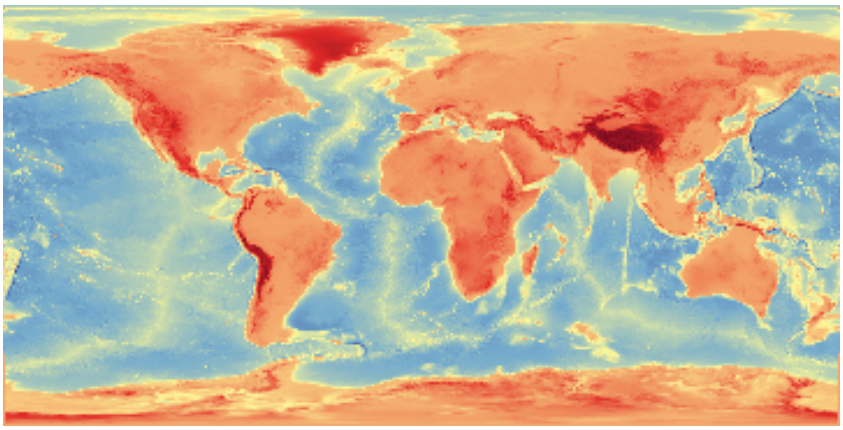
The package include several colour palettes which can be used for plotting purposes. An additional palette option developed specifically with DEMs in mind is *earth*. This combines the *ocean* and *terra* palettes and automatically sets the breaks to differentiate between marine and terrestrial cells.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

```
showPal()
```

3.2 RasterLayer

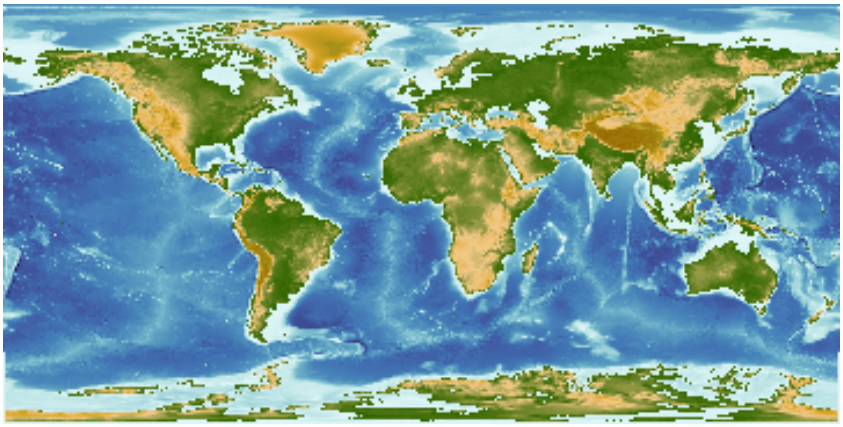
The `mapplot()` function for RasterLayer works similar to the `plot()` function. The `mapplot` function for the Raster* objects include a default palette, omits the legend, axes and the bounding box.

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



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

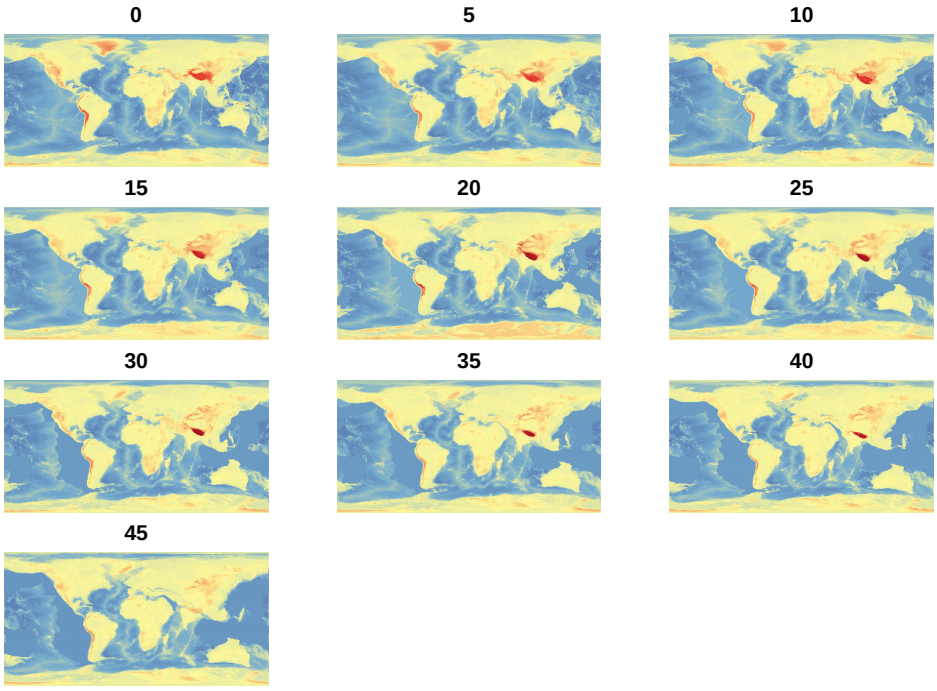
Using the earth palette



3.3 RasterArray

RasterArrays can be plotted as a multi-faceted plots using `mapplot()`. The `mapplot()` function keeps the structure of the RasterArray in terms of the order that the plots are generate. 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:** This number of plots titles provided should be equal to the number of layers in the RasterArray.

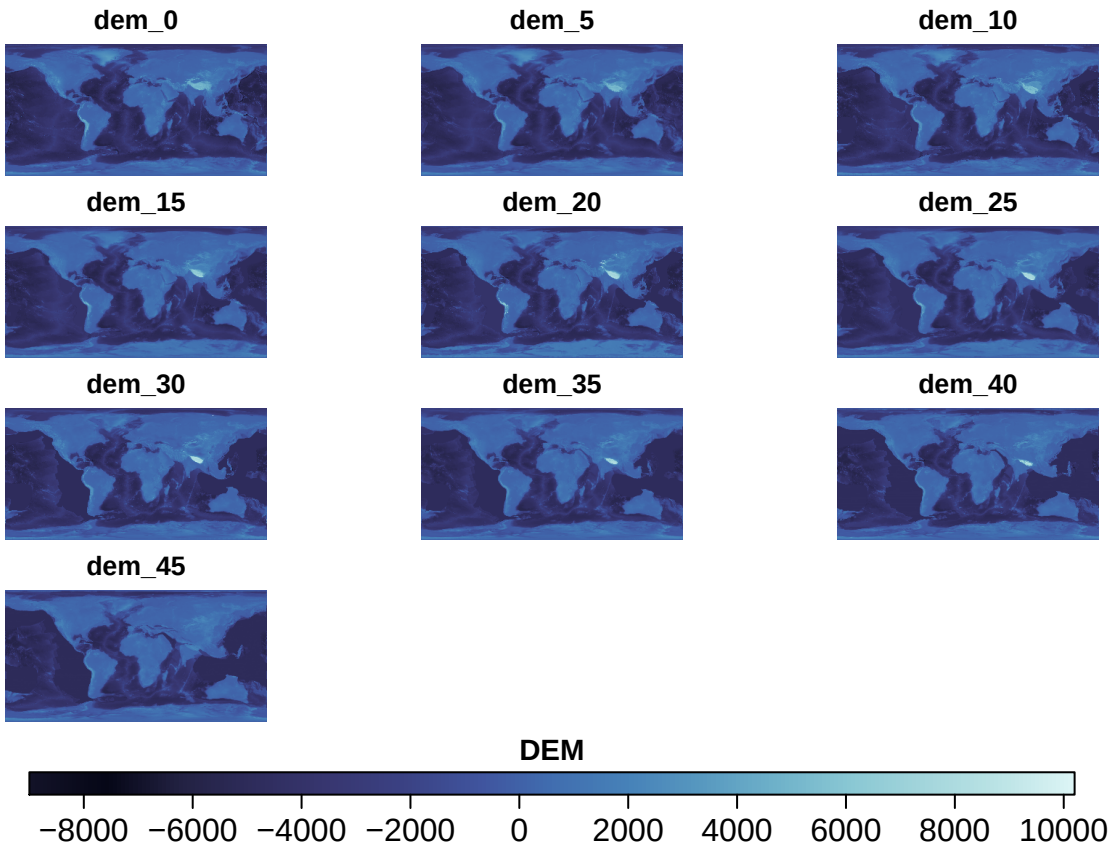
```
data(dems)
mapplot(dems)
```



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))
```



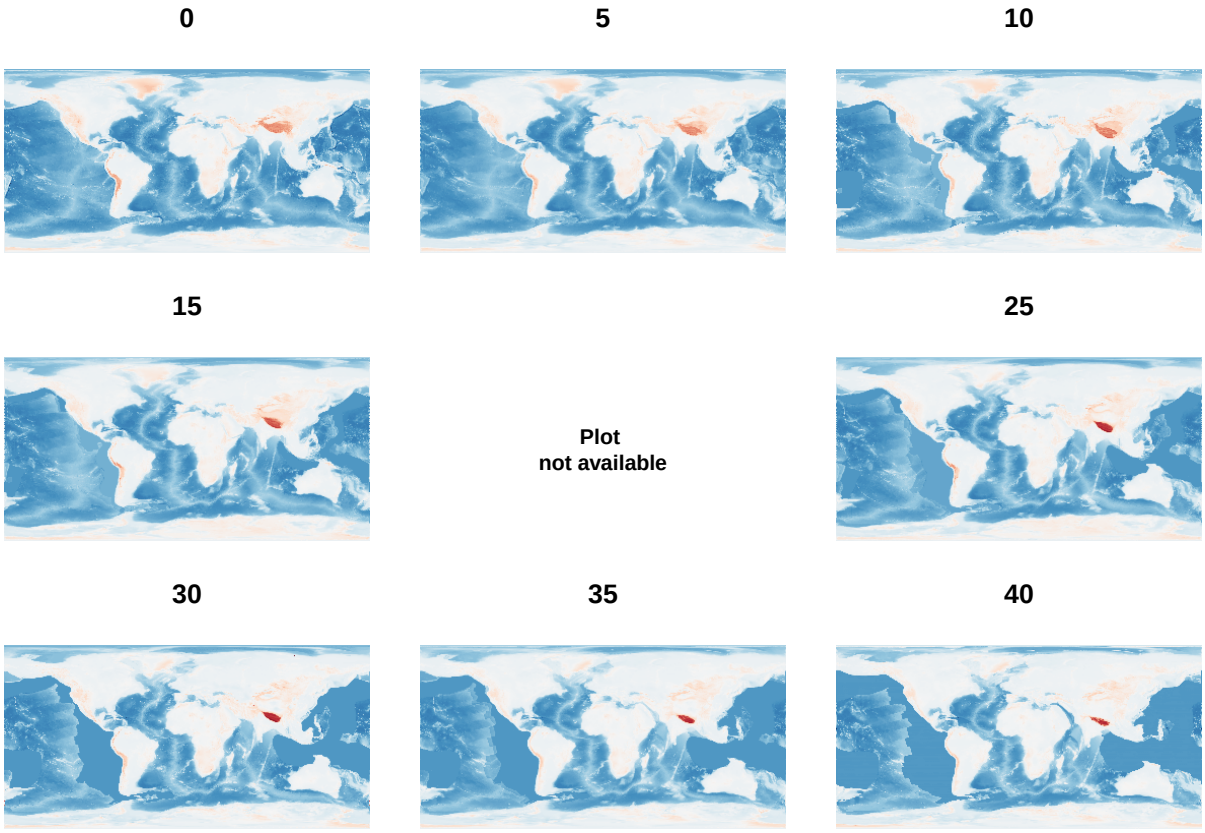
3.3.2 RasterArrays with NA layers

In the event that one of the layers of the dem does not exist i.e. has an NA value instead, this plot corresponding to the 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
```

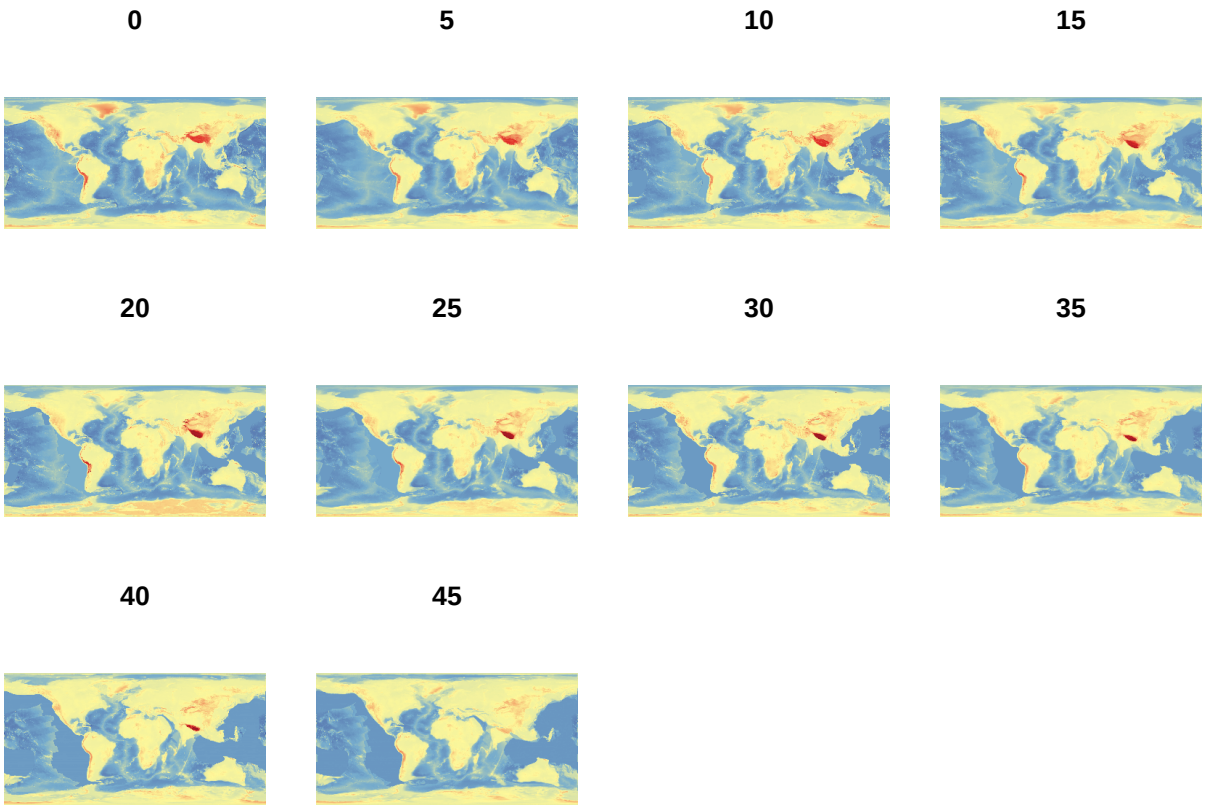
```
mapplot(dems, col="coldhot")
```



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)
```



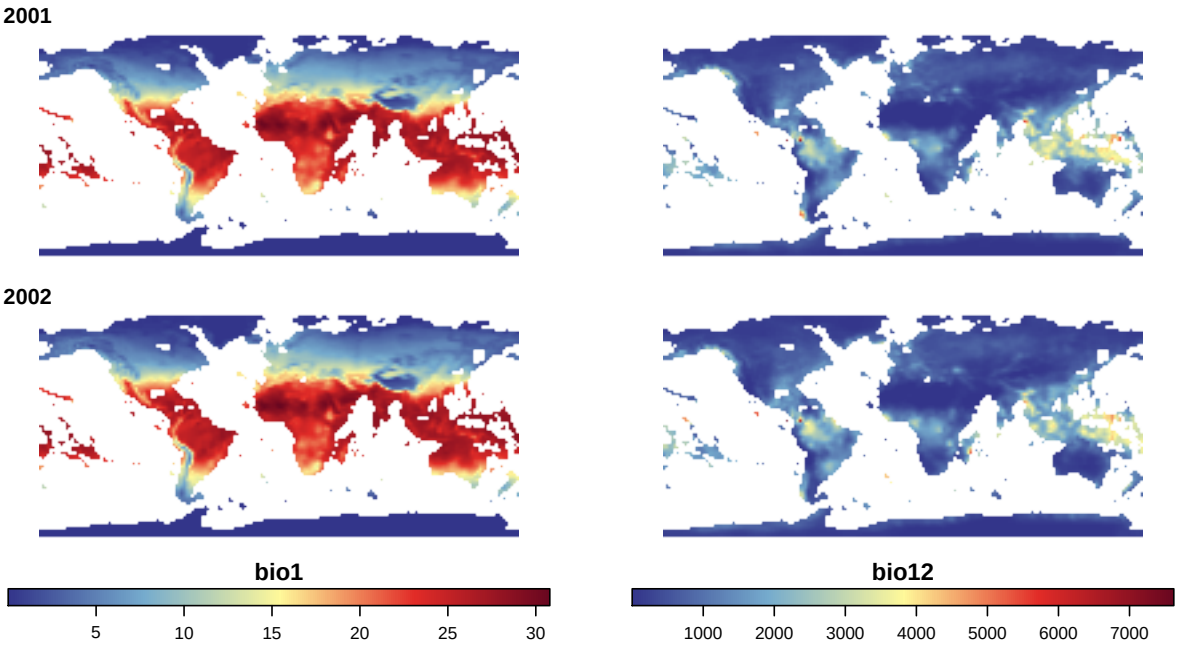
Sometimes, there might be too many plots for them to all plot on one single page. In this instance, the argument `multi` can be set to `TRUE` to allow the figures to be plotted on multiple figures.

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

3.3.4 Multiple variables in RasterArray

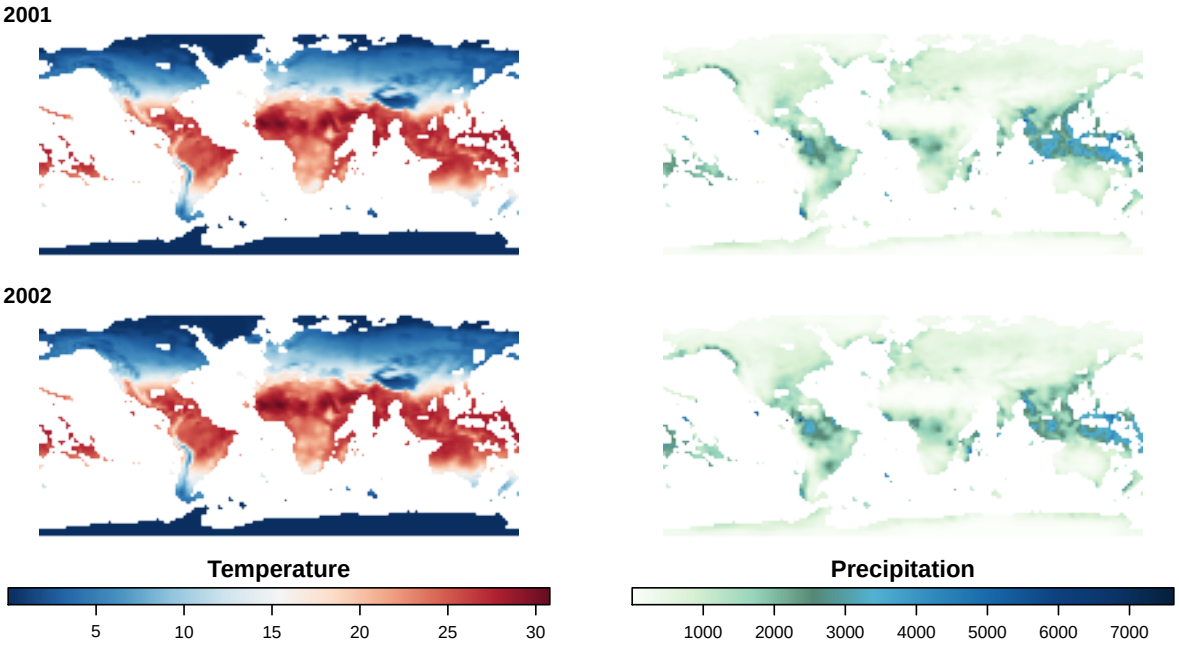
The above described methods can also be applied to RasterArrays containing more than one variables. However, in this instance, the number of columns automatically defaults to the number of variables in the array. Also, plot titles for each individual plot is not printed. Instead, row labels corresponding to the row names of each variable is added and when `legend=TRUE` the default legend title defaults to the variables names in the RasterArray.

```
data(clim)

mapplot(clim[1:2,], legend=TRUE)
```



The row labels and legend title can be edited by using the argument `rowlabels` and `legend.title` respectively. Different colours for each variables can also be provided. The number of provided colour palettes should be either 1 or correspond to the number of variables.

```
data(clim)

mapplot(clim[1:2,], col=c("coldhot", "wet"),
        legend=TRUE, legend.title=c("Temperature", "Precipitation"))
```



References

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

Karger, D. N., Conrad, O., Böhner, J., Kawohl, T., Kreft, H., Soria-Auza, R. W., ... Kessler, M. (2017a). Data from: Climatologies at high resolution for the earth's land surface areas. Dryad Digital Repository. https://doi.org/10.5061/dryad.kd1d4

Karger, D. N., Conrad, O., Böhner, J., Kawohl, T., Kreft, H., Soria-Auza, R. W., ... Kessler, M. (2017b). Climatologies at high resolution for the earth's land surface areas. Scientific Data, 4(1), 170122. https://doi.org/10.1038/sdata.2017.122

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/