ACCESSING ERA-Interim DATA FROM THE SantanderMetGroup DATA SERVER

SantanderMetGroup

29 Dec 2014

Contents

1 Obtaining access authorization

The following document briefly describes the steps to download ERA-Interim data from the SatanderMetGroup THREDDS Data Server for downscaling purposes within the R environment.

As different terms-of-use and policies apply to the different datasets stored in the SantanderMetGroup THREDDS Data Server, a fine-grained user authorization scheme has been implemented using the THREDDS Administration Panel (TAP), which allows user registration and data access authorization for a number of EU-funded projects and initiatives, including the ERA-Interim dataset exposed for downscaling purposes in the framework of the Cost Action VALUE.

In all cases, dataset access is conditioned to the acceptance of the particular usage terms and conditions. Further instructions on TAP registration are provided in this link, provided that in this particular case the group selection corresponds to the **VALUE** project.

2 Accessing data from R

The R package `downscaleR` is envisaged as a user-friendly tool for climate data pre-processing, access and manipulation, and also further post-processing and analytical procedures including bias correction and *perfect-prog* downscaling. Package `downscaleR` provides the necessary functions for accessing and adequately post-processing the ERA-Interim data (interpolation, EOF analysis. . .) for downscaling purposes.

3 Installing *downscaleR*

To obtain the latest stable release of the `downscaleR` package, it is recommended to use the `devtools` utility to install packages directly from the GitHub repo. To this aim, first check that package `devtools` is installed on your system, otherwise install it:

```r
if (!require("devtools")) install.packages("devtools")
```

Then, the `downscaleR` package and its dependencies are installed following the adequate order by entering the following instruction into the R console:

```r
devtools::install_github(c("SantanderMetGroup/downscaleR.java@stable",
"SantanderMetGroup/downscaleR@stable"))
```

NOTE: The `downscaleR` package relies on the powerful capabilities of the Unidata's netCDF-Java API, called from R via the `rJava` package. Thus, the Java Runtime Environment need to be installed and adequately configured to be found by R. If problems arise during the `rJava` installation, please refer to this link.

---

4 Accessing data

4.1 TAP login

Once a valid username and a password are obtained from the TAP, providing authorization for accessing ERA-Interim, login can be performed from R using the `login_TAP` function (see the help for details in case of proxy connections).

```r
library(downscaleR)
```

```
## Loading required package: rJava
## Loading required package: downscaleR.java
## NetCDF Java Library v4.3.22 (27 May 2014) loaded and ready
## Loading required package: maps
## downscaleR version 0.5-1 (29-Dec-2014) is loaded
```

```r
login_TAP(username = "user", password = "9999a")
```

4.2 Dataset overview

The ERA-Interim dataset is exposed in the following URL: http://meteo.unican.es/tds5/catalogs/interim/interimReanalysisDatasets.html?dataset=interim/daily/interim20_daily.ncml

The OpenDAP access is given through the following URL: http://meteo.unican.es/tds5/dodsC/interim/daily/interim20_daily.ncml

This is used as input argument for the inventory and loading functions, as described next:

A quick overview of the available variables in the dataset can be obtained via the `dataInventory` function:

```
dataset <- "http://meteo.unican.es/tds5/dodsC/interim/daily/interim20_daily.ncml"
di <- dataInventory(dataset)
```

```
## [2014-12-29 09:40:24] Doing inventory ...
## [2014-12-29 09:40:28] Retrieving info for 'Q250' (30 vars remaining)
## [2014-12-29 09:40:30] Retrieving info for 'T250' (29 vars remaining)
## [2014-12-29 09:40:30] Retrieving info for 'U250' (28 vars remaining)
## [2014-12-29 09:40:30] Retrieving info for 'V250' (27 vars remaining)
## [2014-12-29 09:40:30] Retrieving info for 'Z250' (26 vars remaining)
## [2014-12-29 09:40:30] Retrieving info for 'Q500' (25 vars remaining)
## [2014-12-29 09:40:30] Retrieving info for 'T500' (24 vars remaining)
## [2014-12-29 09:40:30] Retrieving info for 'U500' (23 vars remaining)
## [2014-12-29 09:40:30] Retrieving info for 'V500' (22 vars remaining)
## [2014-12-29 09:40:30] Retrieving info for 'Z500' (21 vars remaining)
## [2014-12-29 09:40:31] Retrieving info for 'Q700' (20 vars remaining)
## [2014-12-29 09:40:31] Retrieving info for 'T700' (19 vars remaining)
## [2014-12-29 09:40:31] Retrieving info for 'U700' (18 vars remaining)
## [2014-12-29 09:40:31] Retrieving info for 'V700' (17 vars remaining)
## [2014-12-29 09:40:31] Retrieving info for 'Z700' (16 vars remaining)
## [2014-12-29 09:40:31] Retrieving info for 'Q850' (15 vars remaining)
## [2014-12-29 09:40:31] Retrieving info for 'T850' (14 vars remaining)
## [2014-12-29 09:40:31] Retrieving info for 'U850' (13 vars remaining)
## [2014-12-29 09:40:31] Retrieving info for 'V850' (12 vars remaining)
## [2014-12-29 09:40:32] Retrieving info for 'Z850' (11 vars remaining)
## [2014-12-29 09:40:32] Retrieving info for 'Q1000' (10 vars remaining)
## [2014-12-29 09:40:32] Retrieving info for 'T1000' (9 vars remaining)
## [2014-12-29 09:40:32] Retrieving info for 'U1000' (8 vars remaining)
## [2014-12-29 09:40:32] Retrieving info for 'V1000' (7 vars remaining)
## [2014-12-29 09:40:32] Retrieving info for 'Z1000' (6 vars remaining)
## [2014-12-29 09:40:32] Retrieving info for 'MN2T' (5 vars remaining)
## [2014-12-29 09:40:32] Retrieving info for 'MX2T' (4 vars remaining)
## [2014-12-29 09:40:32] Retrieving info for 'SLP' (3 vars remaining)
## [2014-12-29 09:40:33] Retrieving info for 'TP' (2 vars remaining)
## [2014-12-29 09:40:33] Retrieving info for '2T' (1 vars remaining)
## [2014-12-29 09:40:33] Retrieving info for 'SST' (0 vars remaining)
## [2014-12-29 09:40:33] Done.
```

The printed information on screen indicates that the ERA-Interim dataset is composed of 30 variables:

```
names(di)
```

```
##  [1] "Q250"  "T250"  "U250"  "V250"  "Z250"  "Q500"  "T500"  "U500"
##  [9] "V500"  "Z500"  "Q700"  "T700"  "U700"  "V700"  "Z700"  "Q850"
## [17] "T850"  "U850"  "V850"  "Z850"  "Q1000" "T1000" "U1000" "V1000"
## [25] "Z1000" "MN2T"  "MX2T"  "SLP"   "TP"    "2T"    "SST"
```

Particular information for any of the variables can be obtained printing the corresponding element. For instance, the characteristics of the geopotential at 500 mb isobaric surface pressure level (variable `Z500`) are next shown:

```r
str(di$Z500)
```

```
## List of 4
##  $ Description: chr "Geopotential"
##  $ DataType   : chr "float"
##  $ Units      : chr "m**2 s**-2"
##  $ Dimensions :List of 3
##   ..$ time:List of 4
##   .. ..$ Type      : chr "Time"
##   .. ..$ TimeStep  : chr ".99919 days"
##   .. ..$ Units     : chr "days since 1950-01-01 00:00:00"
##   .. ..$ Date_range: chr "1979-01-01T12:00:00Z - 2012-12-31T12:00:00Z"
##   ..$ lat :List of 3
##   .. ..$ Type  : chr "Lat"
##   .. ..$ Units : chr "degrees north"
##   .. ..$ Values: num [1:91] -90 -88 -86 -84 -82 -80 -78 -76 -74 -72 ...
##   ..$ lon :List of 3
##   .. ..$ Type  : chr "Lon"
##   .. ..$ Units : chr "degrees east"
##   .. ..$ Values: num [1:181] -180 -178 -176 -174 -172 -170 -168 -166 -164 -162 ...
```

4.3 Data download

Data download is straightforward using the `loadGridData` function. Argument names are intuitive, and the function has been conceived thinking in the needs of the downscaling community, being simple to obtain spatio-temporal slices for particular spatial domains, seasons and years.

4.3.1 Single variable downloads

Suppose we are interested in one single variable for a particular domain centered on the Iberian Peninsula and France, for instance winter (DJF) geopotential at 500 mb for the period 1981-2000:

```r
z500 <- loadGridData(dataset, var = "Z500", dictionary = FALSE, lonLim = c(-10,10),
             latLim = c(34,50), season = c(12,1,2), years = 1981:2000)
```

```
## [2014-12-29 09:42:55] Defining geo-location parameters
## [2014-12-29 09:42:59] Defining time selection parameters
## [2014-12-29 09:42:59] Retrieving data subset ...
## [2014-12-29 09:43:19] Done
```

The R object `z500` represents what we call a **field** in `downscaleR`. A field is essentially a `list`, whose elements contain all the necessary information to work with the data.

```r
str(z500)
```

```
## List of 4
##  $ Variable:List of 3
##   ..$ varName   : chr "Z500"
##   ..$ isStandard: logi FALSE
##   ..$ level     : NULL
##  $ Data    : num [1:1805, 1:9, 1:11] 56008 55603 55774 55721 55922 ...
##   ..- attr(*, "dimensions")= chr [1:3] "time" "lat" "lon"
##  $ xyCoords:List of 2
##   ..$ x: num [1:11] -10 -8 -6 -4 -2 0 2 4 6 8 ...
##   ..$ y: num [1:9] 34 36 38 40 42 44 46 48 50
##   ..- attr(*, "projection")= chr "LatLonProjection"
##  $ Dates   :List of 2
##   ..$ start: chr [1:1805] "1980-12-01 12:00:00 GMT" "1980-12-02 12:00:00 GMT" "1980-12-03 12:00:00 G
##   ..$ end  : chr [1:1805] "1980-12-02 12:00:00 GMT" "1980-12-03 12:00:00 GMT" "1980-12-04 12:00:00 G
##  - attr(*, "dataset")= chr "http://meteo.unican.es/tds5/dodsC/interim/daily/interim20_daily.ncml"
```

Climate data requested are stored in the `$Data` element of the list. It is a N-dimensional array, where N is the number of dimensions depending on the type of data request. In this case, the vertical dimension is dropped as it is of length one (500 mb), so the returned array is 3-dimensional, with the dimension names stored as an attribute named `dimensions`:

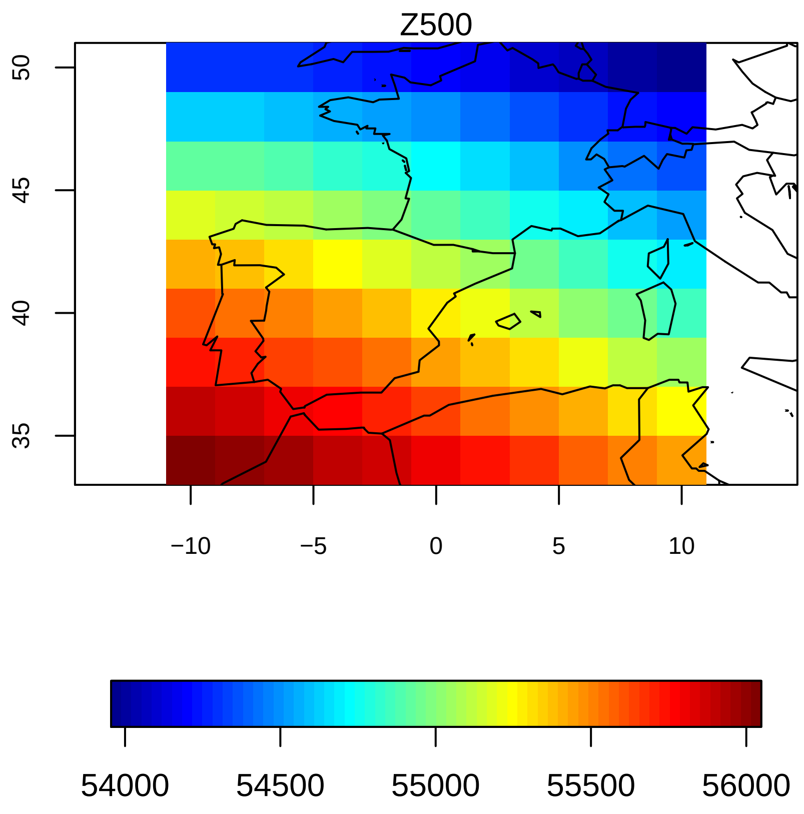
```r
str(z500$Data)
```

```
##  num [1:1805, 1:9, 1:11] 56008 55603 55774 55721 55922 ...
##  - attr(*, "dimensions")= chr [1:3] "time" "lat" "lon"
```

The dimensions are always arranged in their *canonical* order, this is: member > time > lat > lon. The help of the function provides further details.

Finally, a visual representation of the data loaded can be obtained using the `plotMeanField` function, that represents the temporal mean of the data loaded (i.e., mean winter Z500 for the period 1981-2000):

```r
plotMeanField(z500)
```



4.3.2 Loading several predictors simultaneously

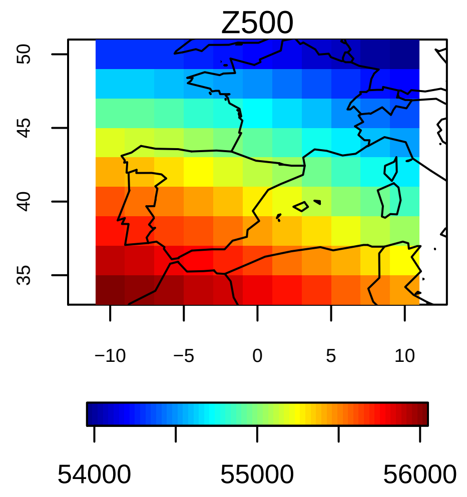
Usually in downscaling application we use more than one predictor (or *field*). Several predictors (a **multifield**) can be simultaneously loaded for a common spatio-temporal domain using the `loadMultiField` function. The function also allows for the definition of a particular grid different from the native grid of the data, thus enabling data loading and interpolation in a single step. To this aim, the optional arguments `new.grid`, defining the new output regular grid and `interp.method`, indicating the interpolation method, can be indicated.

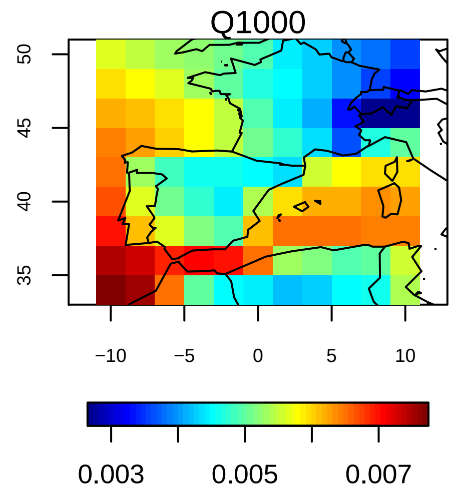
In this example, we load three predictors: geopotential at 500 mb, specific humidity at 1000 mb and sea-level pressure, returning a multifield. We preserve the original grid definition, thus ommitting the interpolation step:

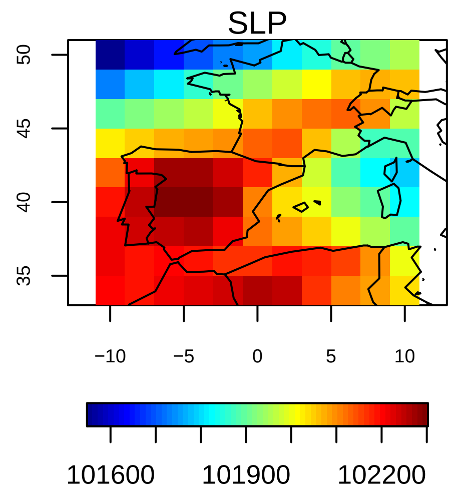
```
mf <- loadMultiField(dataset, vars = c("Z500", "Q1000", "SLP"), dictionary = FALSE,
            lonLim = c(-10, 10), latLim = c(34,50), season = c(12,1,2), years = 1981:2000)
```

```
## [2014-12-29 09:55:58] Loading predictor 1 (Z500) out of 3
## [2014-12-29 09:56:31] Loading predictor 2 (Q1000) out of 3
## [2014-12-29 09:57:00] Loading predictor 3 (SLP) out of 3
## [2014-12-29 09:57:28] Using the original grid as no 'new.grid' has been introduced
## [2014-12-29 09:57:28] Done.
```

```
plotMeanField(mf)
```







In this case, the `$Data` array of the `mf` object has an additional dimension corresponding to each of the variables composing the multifield (dimension `var`):

```
str(mf$Data)
```

```
##  num [1:3, 1:1805, 1:9, 1:11] 5.60e+04 5.34e-03 1.02e+05 5.56e+04 4.98e-03 ...
##  - attr(*, "dimensions")= chr [1:4] "var" "time" "lat" "lon"
```

5 Basic data manipulation

Some of the typical operations involving downscaling applications are straightforward in `downscaleR`:

5.1 Interpolation

Currently, nearest-neighbour and bilinear methods are implemented for interpolation via the `interpGridData` function. The function `getGrid` can be useful for obtaining the grid definition fo a previously existing object in an adequate format for passing it to the interpolator.

```r
getGrid(z500)
```

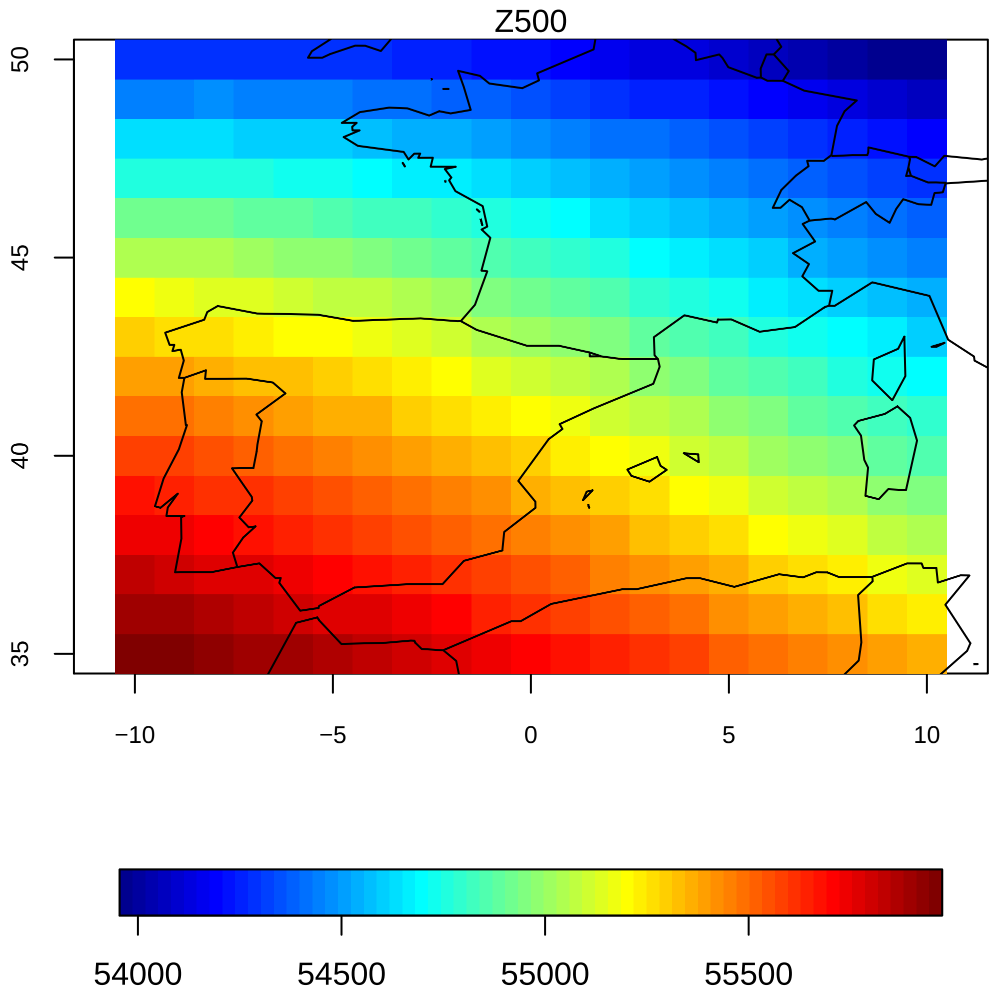
```
## $x
## [1] -10  10
##
## $y
## [1] 34 50
##
## attr(,"projection")
## [1] "LatLonProjection"
## attr(,"resX")
## [1] 2
## attr(,"resY")
## [1] 2
```

Interpolation can be applied directly to a field:

```r
z500.1deg <- interpGridData(z500, new.grid = list(x = c(-10,10,1), y = c(35,50,1)),
               method = "bilinear")
```

```
## [2014-12-29 10:08:28] Performing bilinear interpolation... may take a while
## [2014-12-29 10:08:39] Done
```

```r
plotMeanField(z500.1deg)
```

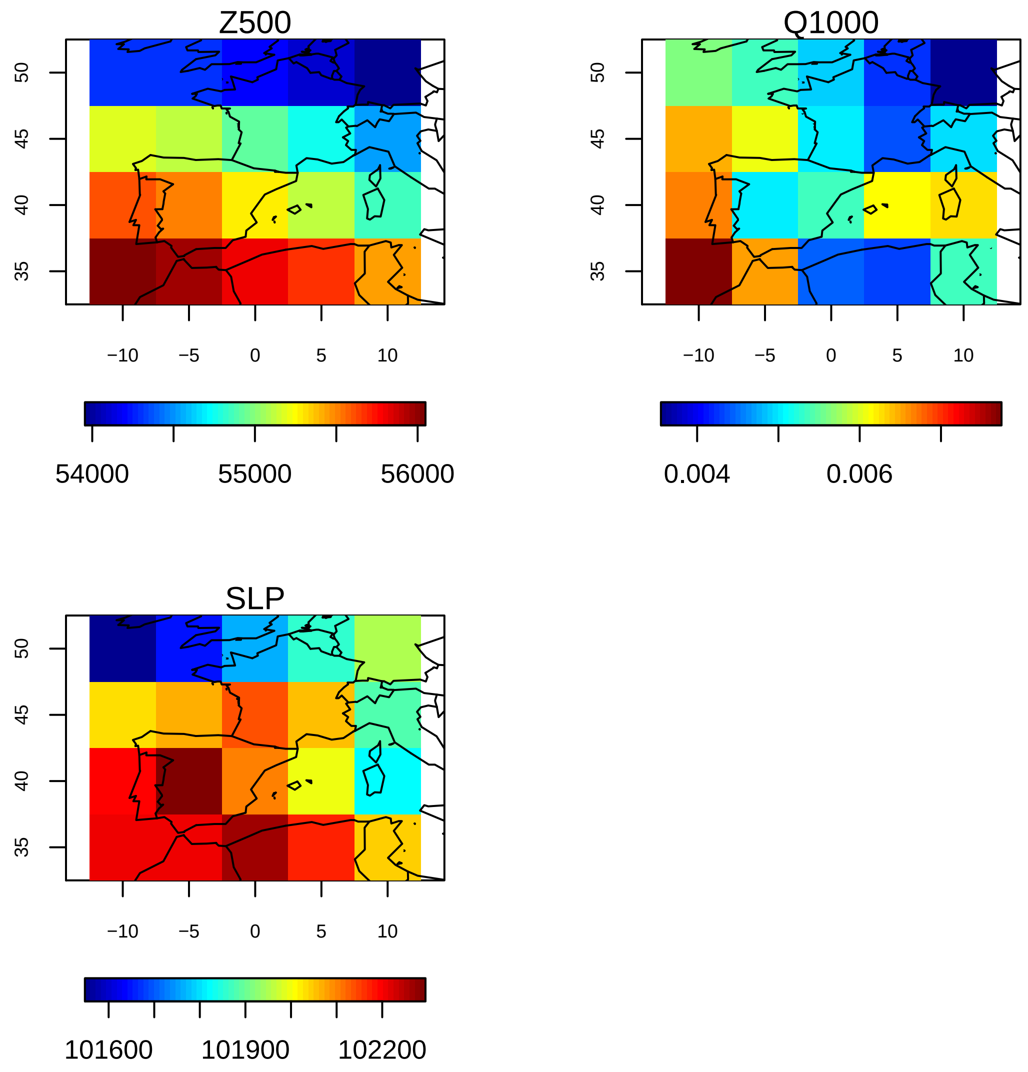


In the case of multifields, the interpolation must be indicated at the loading step. For instance, for a regular grid of 5-degree resolution in X and Y using nearest neighbour:

```r
mf.5deg <- loadMultiField(dataset, vars = c("Z500", "Q1000", "SLP"), dictionary = FALSE,
             lonLim = c(-10, 10), latLim = c(34,50), season = c(12,1,2), years = 1981:2000,
             new.grid = list(x = c(-10, 10, 5), y = c(35, 50, 5)), interp.method = "nearest")
```

```
## [2014-12-29 10:35:58] Loading predictor 1 (Z500) out of 3
## [2014-12-29 10:36:32] Loading predictor 2 (Q1000) out of 3
## [2014-12-29 10:37:00] Loading predictor 3 (SLP) out of 3
## [2014-12-29 10:37:28] Regridding variable 1 (Z500) out of 3 ...
## [2014-12-29 10:37:28] Regridding variable 2 (Q1000) out of 3 ...
## [2014-12-29 10:37:29] Regridding variable 3 (SLP) out of 3 ...
## [2014-12-29 10:37:29] Done.
```

```r
plotMeanField(mf.5deg)
```



After interpolation a particular attribute is assigned to the `$xyCoords` element of the field or multifield:

```r
attributes(z500.1deg$xyCoords)
```

```
## $names
## [1] "x" "y"
##
## $projection
## [1] "LatLonProjection"
##
## $interpolation
## [1] "bilinear"
##
## $resX
## [1] 1
##
## $resY
## [1] 1
```

```r
attributes(mf.5deg$xyCoords)
```

```
## $names
## [1] "x" "y"
##
## $projection
## [1] "LatLonProjection"
##
## $interpolation
## [1] "nearest"
##
## $resX
## [1] 5
##
## $resY
## [1] 5
```

5.2 Principal Component Analysis

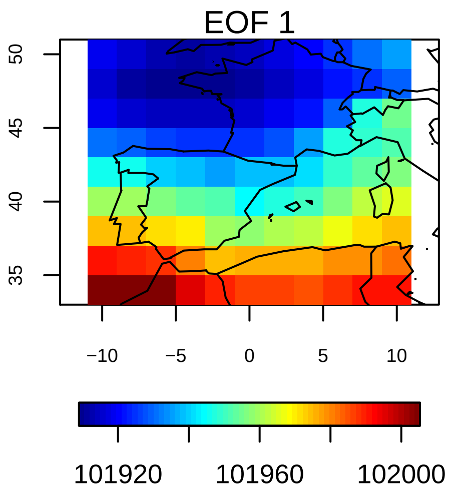
It is common practice the application of data dimensionality reduction techniques like principal component (EOF, empirical orthogonal functions) analysis. The use of EOFs provides a number of advantages against the use of the raw predictors alone. In `downscaleR`, the PCA/EOF analysis is undertaken by the `prinComp` function and related methods (`fieldFromPCs`, `plotEOF`...), that we briefly describe in the following:

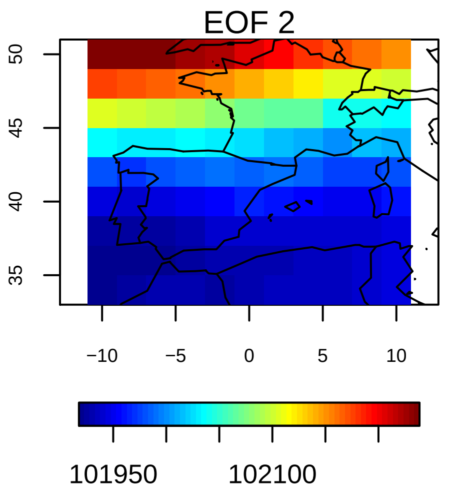
Principal components explaining the 95% of total variance:

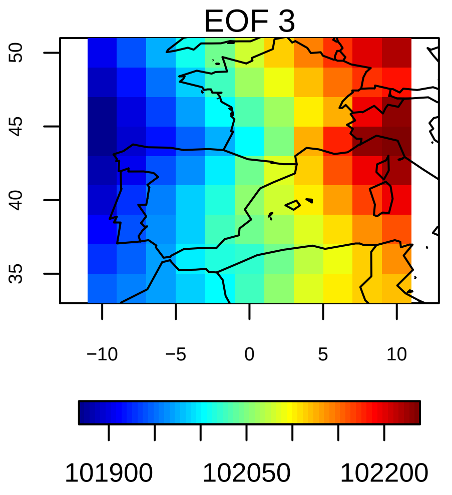
```r
pca1 <- prinComp(mf, v.exp = .95)
```

```
## [2014-12-29 11:12:36] Performing PC analysis on 3 variables plus their combination...
## [2014-12-29 11:12:36] Done
```

```r
plotEOF(pca1, var = "SLP")
```





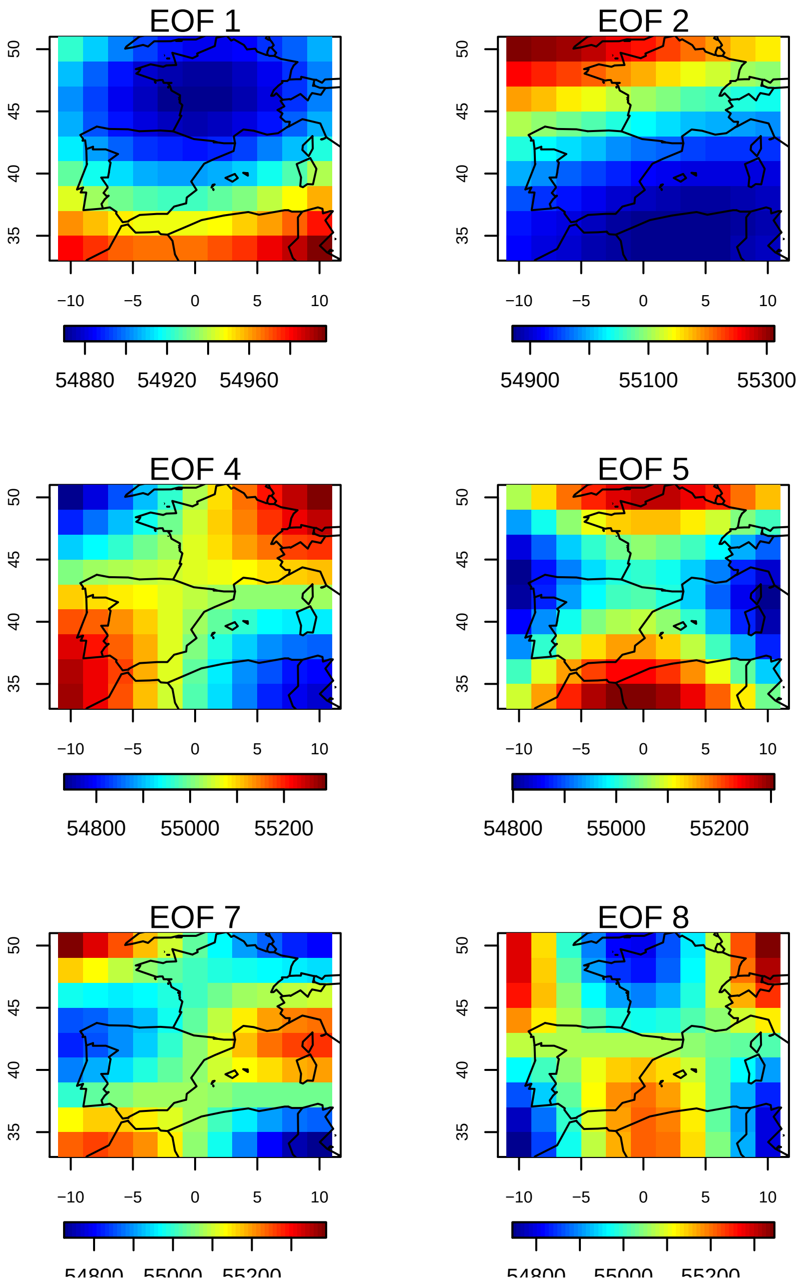


The 8 first principal components:

```
pca2 <- prinComp(mf, n.eofs = 8)
```

```
## [2014-12-29 11:12:37] Performing PC analysis on 3 variables plus their combination...
## [2014-12-29 11:12:37] Done
```

```
plotEOF(pca2, var = "Z500")
```



The object returned by `prinComp` can be directly passed to the different bias correction and downscaling functions in `downscaleR`. See for instance this example.