The following examples are partly based on locally stored data. Only the `loadSystem4` function does not require locally stored datasets because it works by remotely accessing the System4 databases stored in the SPECS-EUPORIAS Data Server. Therefore, it is recommended that the **meteoR.zip** file is downloaded and unzipped in a convenient directory. Once the zip file has been downloaded and unzipped, we will define the 'r' folder as the working directory. Therefore, in the following examples all the path names given will be relative to the 'r' directory.

The following expression lists all the R functions available, read them and loads them into the R working session:

```
> rfuncs <- list.files(pattern = "\\.R$")
> print(rfuncs)
[1] "dataInventory.R"    "loadData.R"         "loadObservations.R" "loadSystem4.R"      "makeNcmlDataset.R"  "makeVocabula
> for (i in 1:length(rfuncs)) {
+      source(rfuncs[i])
+ }
```

loadSystem4

In the next lines we describe an illustrative example of the `loadSystem4` function. We will retrieve System4 simulation data for the Iberian Peninsula, considering mean surface temperature for January and the first simulation member, for the 10-year period 1990-1999. This simple example has been chosen because of the fast data access (note that this also depends on the connection speed). Using a standard broadband connection, loading the following dataset took approximately 19 seconds:

```
> openDAP.query <- loadSystem4(dataset = "http://www.meteo.unican.es/tds5/dodsC/system4/System4_Seasonal_15Members.ncml",
+                              var = "tas", members = 1,
+                              lonLim = c(-10,5), latLim = c(35,45),
+                              season = 1, years = 1990:1999, leadMonth = 1)
```

Data are now ready for analysis into our R session:

```
> str(openDAP.query)
List of 7
$ VarName      : chr "Mean_temperature_at_2_metres"
$ VarUnits     : chr "degC"
$ TimeStep     :Class 'difftime'  atomic [1:1] 1
 .. ..- attr(*, "tzone")= chr ""
 .. ..- attr(*, "units")= chr "days"
$ MemberData   :List of 1
 ..$ : num [1:310, 1:280] 13.3 13.9 12.5 13 13 ...
$ LatLonCoords : num [1:280, 1:2] 45 44.2 43.5 42.7 42 ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : NULL
 .. ..$ : chr [1:2] "lat" "lon"
$ RunDates     : POSIXlt[1:310], format: "1989-12-01" "1989-12-01" "1989-12-01" ...
$ ForecastDates:List of 2
 ..$ Start: POSIXlt[1:310], format: "1990-01-01" "1990-01-02" "1990-01-03" ...
 ..$ End  : POSIXct[1:310], format: "1990-01-02" "1990-01-03" "1990-01-04" ...
```

A common task consists of the representation of data, e.g. by mapping the spatial mean for the period considered. Another common task is the representation of time series for selected point locations/grid cells. In this example, we will map the mean temperature field for the period selected (1990-99) preserving the original spatial resolution of the model. Furthermore, we will display time series of the requested dataset at four grid points coincident with the locations of four Spanish cities. To this aim, we will make use of some `base` R functions and also from some contributed packages that can be very useful for climate data handling and representation.

```
> # Calculation of the mean values of the period for each grid cell:
> mean.field <- colMeans(openDAP.query$MemberData[[1]])
> # Creation of a matrix with selected point locations:
> city.names <- c("Sevilla", "Madrid", "Santander", "Zaragoza")
> locations <- matrix(c(-5.9, 37.4167, -3.68, 40.4, -3.817, 43.43, -0.8167, 41.667), ncol=2, byrow = TRUE)
> dimnames(locations) <- list(city.names, c("lon","lat"))
> print(locations)
            lon     lat
```

```
Sevilla   -5.9000 37.4167
Madrid    -3.6800 40.4000
Santander -3.8170 43.4300
Zaragoza  -0.8167 41.6670
```

Note that the geographical coordinates of the requested spatial domain are nor perfectly uniform, and as a result it is not possible to represent the data as a regular grid. To overcome this problem, we perform an interpolation, although we preserve the native grid cell size of the model (0.75deg) for data representation. The R package ?akima provides a extremely fast interpolation algorithm by means of the `interp` function.

```
> lat <- openDAP.query$LatLonCoords[ ,1]
> lon <- openDAP.query$LatLonCoords[ ,2]
# Requires "akima::interp" for regular grid (bilinear) interpolation
> library(akima)
> grid.075 <- interp(lon, lat, mean.field, xo = seq(min(lon), max(lon), .75), yo = seq(min(lat), max(lat), .75))
```

After the interpolation, data are now in a regular grid. Note that the object `grid.075` is a list with the usual `x,y,z` components as required by many R functions for gridded data representation (e.g. `image`, `contour`, `persp`...)

```
> str(grid.075)
List of 3
$ x: num [1:19] -9.75 -9 -8.25 -7.5 -6.75 ...
$ y: num [1:14] 35.2 36 36.7 37.5 38.2 ...
$ z: num [1:19, 1:14] 15.1 15.1 15 15 14.3 ...
```

In the following lines of code we plot the mean temperature field. In addition, we will also add to the map the point locations of the selected cities for which the time series will be represented. The R package ?fields provides many useful tools for spatial data handling and representation, including a world map that can be easily incorporated in our plots.

```
> # Representation of the mean temperature of the period
> library(fields)
> image.plot(grid.075, asp=1,ylab = "latitude", xlab = "longitude", col=topo.colors(36),
+           main = "Mean surface T January 1990-99", legend.args=list(text="degC", side=3, line=1))
# Adds selected locations to the plot and puts labels:
> points(locations, pch=15)
> text(locations, pos=3, city.names)
# Adds the world map to the current plot:
> world(add=TRUE)
```

Next, we plot the time series for the selected locations. To this aim, we calculate the nearest grid points to the specified locations. This can be easily done using the function `fields::rdist`. Note that the output of `loadSystem4` returns a matrix of Lat-Lon coordinates, as usually found in many climate datasets. However, the usual format of 2D coordinates matrix in R is Lon-Lat. As a result, note that we specify the coordinates by reversing the column order (i.e.: `openDAP.query$LatLonCoords[ ,2:1]` instead of `openDAP.query$LatLonCoords`):

```
> # Creation of a Euclidean distance matrix among all pairs of selected locations and grid points
> dist.matrix <- rdist(openDAP.query$LatLonCoords[ ,2:1], locations)
> # Positions of the nearest grid points
> index <- rep(NA,ncol(dist.matrix))
> for (i in 1:ncol(dist.matrix)) {
+      index[i] <- which.min(dist.matrix[ ,i])
+ }
> # index contains the column positions in the `MemberData` matrices
> locations.data <- openDAP.query$MemberData[[1]][ ,index]
> colnames(locations.data) <- city.names
> str(locations.data)
num [1:310, 1:4] 7.71 8.78 10.77 10.88 11.55 ...
- attr(*, "dimnames")=List of 2
 ..$ : NULL
 ..$ : chr [1:4] "Sevilla" "Madrid" "Santander" "Zaragoza"
```

The object `locations.data` is a matrix in which time series are arranged in columns for each of the four locations selected.

```
> ylimits <- c(floor(min(locations.data)), ceiling(max(locations.data)))
> plot(locations.data[ ,1], ty='n', ylim = ylimits, axes=FALSE, ylab="degC", xlab="Year")
> axis(1,at = seq(1,31*11,31), labels=c(1990:1999,""))
> axis(2, ylim=ylimits)
> abline(v=seq(1,31*10,31), lty=2)
> for (i in 1:ncol(locations.data)) {
+      lines(locations.data[ ,i], col=i)
+ }
> legend("bottomleft", city.names, lty=1, col=1:4)
> title(main = "Mean surface Temperature January")
```

loadObservations

Loading Observations

The function `loadObservations` is intended to deal with observational datasets from weather stations stored as csv files in a standard format. In the directory "./datasets/observations/Iberia_ECA" there is an example dataset.

```
> list.files("./datasets/observations/Iberia_ECA")
[1] "ecaIberia.nc" "Master.csv"   "pr.csv"       "tas.csv"      "tasmax.csv"   "tasmin.csv"
```

As we can see, there is a number of files with the name of the variables they store, and a Master.csv file, which contains the required metadata in order to identify each station. This is how the Master.csv file looks like:

```
> master <- read.csv("./datasets/observations/Iberia_ECA/Master.csv")
> str(master)
'data.frame':      28 obs. of  5 variables:
     $ Id       : int  33 229 230 231 232 233 234 236 309 336 ...
$ Longitude: num  1.38 -6.83 -3.68 -4.49 -4.01 ...
$ Latitude : num  43.6 38.9 40.4 36.7 40.8 ...
$ Altitude : int  151 185 667 7 1894 790 251 44 43 704 ...
$ Metadata : Factor w/ 1 level " Data provided by the ECA&D project. Available at http://www.ecad.eu": 1 1 1 1 1 1 1 1 1 1
```

The Master contains the following fields:

- `Id`: Identification code of the station. This code is used as argument by the function `loadObservations`. Note thas this field should actually be read as a character string, as internally done by the `loadObservations` function. However, in this case `read.csv` by default interpretes it as a numeric value.
- `Longitude`: longitude
- `Latitude`: latitude
- `Altitude`: altitude
- `Metadata`: other metadata associated to the dataset

First of all we will plot the stations so that we can get an idea of their geographical situation and extent:

```
> library(fields)
> plot(master[ ,2:3], asp=1, pch=15, col="red")
> world(add=TRUE)
```

In order to get a vector with the correct IDs as character strings instead of numeric values, we can load again the corresponding column using the argument `colClasses = "character"`

```
> stationIDs <- read.csv("./datasets/observations/Iberia_ECA/Master.csv", colClasses = "character")[ ,1]
> stationIDs
[1] "000033" "000229" "000230" "000231" "000232" "000233" "000234" "000236" "000309" "000336" "000414" "000416" "000420" "
```

```
[15] "000788" "000800" "001392" "001398" "003904" "003905" "003907" "003908" "003922" "003928" "003936" "003947" "003948"
```