Wikiprint Book

Title: loadSeasonalForecast

Subject: TracMeteo - udg/ecoms/RPackage/examples

Version: 76

Date: 12/02/2021 07:47:10 AM

Table of Contents

The following examples are partly based on locally stored data. Only the `loadSeasonalForecast` function does not require locally stored datasets because it works by remotely accessing the System4 databases stored in the SPECS-EUPORIAS Data Server (in this case a valid username and password are required though, as detailed ?Authentication section). Therefore, it is recommended that the ?meteoR_v1_0.zip file is downloaded and unzipped in a convenient directory before starting. Please follow the instructions?.

loadSeasonalForecast

In the next lines we describe an illustrative example of the `loadSeasonalForecast` function. **Note that a valid username and password are required** in order to access the SPECS-EUPORIAS Data Server, as detailed ?here. 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 <- loadSeasonalForecast(dataset = "http://www.meteo.unican.es/tds5/dodsC/system4/System4_Seasonal_15Member
+                                       standard.vars = TRUE, dictionary = "datasets/forecasts/System4/System4_Seasonal_
+                                       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. Note that we have set the `standard.vars` argument to `TRUE`, and we have specified a path to the dictionary, a file with extension *.dic*, whose aim is to translate the original variable stored in the dataset (in this case in Kelvin instead of Celsius). More details on the use of the dictionary are provided in this section?.

```
> str(openDAP.query)
List of 6
$ VarName      : chr "tas"
$ isStandard   : logi TRUE
$ 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     : chr [1:310] "1989-12-01" "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" "1990-01-04" ...
 ..$ End  : POSIXlt[1:310], format: "1990-01-02" "1990-01-03" "1990-01-04" "1990-01-05" ...
```

The function has been implemented to access seasonal slices (as determined by the `season` argument. Seasons can be defined in several ways: A single month (e.g. `season = 1` for January, as in this example), a standard season (e.g. `season=1:3` for JFM, or `season=c(12,1,2)` for DJF), or any period of consecutive months (e.g. `season=1:6`, for the first half of the year). Seasons are returned for a given year period (defined by the `years` argument, e.g. `years = 1990:1999` in this example) with a homogeneous forecast lead time (as given by the `leadMonth` argument; in this example `leadMonth = 1` for one-month lead time) with respect to the first month of the selected season. For example, in this particular case the data loaded correspond to the series of January 1990 to January 1999 from the December 1989 to December 1998 runtime forecast. As a result, the length of the time series returned for each of the 280 grid cells is of 310 days (31 days of January * 10 years). Note that it is also possible to work with year-crossing seasons, such as DJF (in this case `season=c(12,1,2)`).

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 at two grid points coincident with the locations of two 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("Madrid", "Santander")
> locations <- matrix(c(-3.68, 40.4, -3.817, 43.43), ncol=2, byrow = TRUE)
```

```
> dimnames(locations) <- list(city.names, c("lon","lat"))
> print(locations)
            lon   lat
Madrid    -3.680 40.40
Santander -3.817 43.43
```

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 `loadSeasonalForecast` 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, 1:2] 2.52 3.57 4.05 6.88 7.36 ...
- attr(*, "dimnames")=List of 2
 ..$ : NULL
 ..$ : chr [1:2] "Madrid" "Santander"
```

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")
> mtext("System4 15 member Seasonal - 1st Member, lead month = 1")
```

Alternatively, for selected point locations the function allows for the retrieval of single-point data. In this case, we can enter the lon and lat coordinates of the desired point in the `lonLim`and `latLim` arguments of the function. The function operates by finding the nearest grid point of the dataset to the given coordinates (in terms of Euclidean distances). For instance, the next two instructions load the data represented in the time series above directly for Santander and Madrid respectively:

```
> santanderData <- loadSeasonalForecast(dataset = "http://www.meteo.unican.es/tds5/dodsC/system4/System4_Seasonal_15Member
+                                       standard.vars = TRUE, dictionary = "datasets/forecasts/System4/System4_Seasonal_15M
+                                       var = "tas", members = 1,
+                                       lonLim = -3.81, latLim = 43.43,
+                                       season = 1, years = 1990:1999, leadMonth = 1)
> madridData <- loadSeasonalForecast(dataset = "http://www.meteo.unican.es/tds5/dodsC/system4/System4_Seasonal_15Members.n
+                                       standard.vars = TRUE, dictionary = "datasets/forecasts/System4/System4_Seasonal_15M
+                                       var = "tas", members = 1,
+                                       lonLim = -3.68, latLim = 40.40,
+                                       season = 1, years = 1990:1999, leadMonth = 1)
> plot(santanderData$MemberData[[1]], ty='l', col = "red")
> lines(madridData$MemberData[[1]], ty='l')
> title("Same data as the previous plot")
```

loadObservations

The function `loadObservations` is intended to deal with observational datasets from weather stations stored as csv files in a standard format. In the directory *meteoR/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
```

The *Master* table 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"
```

We place the labels on top of each location. There are ways to avoid the overlapping of labels in order to explore the dataset, for instance in an interactive fashion by means of the `identify` function .

```
> text(master$Longitude + .2, master$Latitude + .2, stationIDs, cex=.7)
```

In this particular example we are interested in the mean surface temperature from the cities of Santander and Madrid in January. The station codes are *"000230"* and *"001392"* for Madrid and Santander respectively. We will select the period 1990-1999.

```
> stationData <- loadObservations(source.dir="./datasets/observations/Iberia_ECA/", var="tas",
+           standard.vars=FALSE, stationID=c("000230","001392"), startDate="1990-01-01", endDate="1999-12-31")
> str(stationData)
List of 5
$ StationID   : chr [1:2] "000230" "001392"
$ LatLonCoords: num [1:2, 1:2] 40.41 43.46 -3.68 -3.82
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : NULL
 .. ..$ : chr [1:2] "Latitude" "Longitude"
$ Altitude    : int [1:2] 667 64
$ Dates       : POSIXlt[1:3652], format: "1990-01-01" "1990-01-02" "1990-01-03" "1990-01-04" ...
$ Data        : num [1:3652, 1:2] 112 85 90 74 101 109 110 82 86 66 ...
```

```
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : NULL
 .. ..$ : chr [1:2] "000230" "001392"
> ylimits <- c(floor(min(stationData$Data)) * .1, ceiling(max(stationData$Data) * .1))
> plot(stationData$Data[ ,1], ty='n', ylim = ylimits, axes=FALSE, ylab="degC", xlab="Year")
> axis(1,at = seq(1,31*11,31), labels=c(1990:1999,""))
> axis(2, ylim=ylimits)
> abline(v=seq(1,31*10,31), lty=2, col = "grey30")
> for (i in 1:ncol(stationData$Data)) {
+      lines(stationData$Data[ ,i] * .1, col=i)
+ }
> city.names = c("Madrid","Santander")
> legend("topright", paste(city.names, stationData$StationID), lty=1, col=1:2, bty="n")
> title(main = "Mean surface Temperature January - Observations")
```

Creating a dataset

Climate datasets of various types (e.g. reanalysis, RCM/GCM data...) are often stored as collections of netCDF files for each particular variable and/or time slice. These files can be either locally or remotely stored. A convenient way of dealing with this kind of datasets is the use of NcML files. A NcML file is a ?XML representation of netCDF metadata. By means of NcML it is possible to create virtual datasets by modifying and aggregating other datasets, thus providing maximum flexibility and ease of access to data stored in collections of files containing data from different variables/time slices. The function `makeNcmlDataset` is intended to deal with reanalysis, forecasts and other climate data products, often consisting of collections of netCDF files of these characteristics.

In this example, we have chosen several variables commonly used in statistical downscaling applications belonging to the NCEP reanalysis, and stored in different netCDF files. These variables are stored in the following directory:

```
> list.files("./datasets/reanalysis/Iberia_NCEP/")
[1] "Iberia_NCEP.dic"  "NCEP_Q850.nc"     "NCEP_SLPd.nc"     "NCEP_T850.nc"     "NCEP_Z500.nc"
```

Note that the dictionary for this dataset (*Iberia_NCEP.dic*) has also been included in the directory.

The function `makeNcmlDataset` is used to conveniently aggregate the required information so that the inventory/loading functions point to the NcML rather that to the netCDF files. The following call to the function wll create the NcML file in the same directory where the netCDF files are stored:

```
> makeNcmlDataset(source.dir="datasets/reanalysis/Iberia_NCEP/", ncml.file="datasets/reanalysis/Iberia_NCEP/Iberia_NCEP.nc
[2013-05-20 10:00:51]
NcML file "Iberia_NCEP_dataset.ncml" created from 4 files corresponding to 4 variables
Use 'dataInventory(NcML file)' to obtain a description of the dataset
```

The function creates a new NcML file in the directory specified (in this case in the working directory, as no path has been specified), and gives some information about the number of files and variables conforming the dataset. In the next section is described how to find out the different variables stored in the newly created dataset and their characteristics.

dataInventory

With the aid of the `dataInventory`function we can easily retrieve all the necessary information to access and manipulate the variables stored in a dataset. In the following example, we get a description of the NcML dataset created in the previous section, containing several variables of the NCEP reanalysis in the Iberian Peninsula.

```
> inv.iberiaNCEP <- dataInventory("datasets/reanalysis/Iberia_NCEP/Iberia_NCEP.ncml")
# Structure of the inventory
> str(inv.iberiaNCEP)
List of 4
```

```
$ Q   :List of 5
 ..$ Description: chr "Specific humidity"
 ..$ DataType   : chr "float"
 ..$ Units      : chr "kg kg**-1"
 ..$ TimeStep   :Class 'difftime'  atomic [1:1] 24
 .. .. ..- attr(*, "tzone")= chr ""
 .. .. ..- attr(*, "units")= chr "hours"
 ..$ Dimensions :List of 4
 .. ..$ level:List of 3
 .. .. ..$ Type  : chr "Pressure"
 .. .. ..$ Units : chr "millibar"
 .. .. ..$ Values: num 850
 .. ..$ time :List of 3
 .. .. ..$ Type  : chr "Time"
 .. .. ..$ Units : chr "days since 1950-01-01 00:00:00"
 .. .. ..$ Values: POSIXlt[1:16071], format: "1958-01-01" "1958-01-02" "1958-01-03" "1958-01-04" ...
 .. ..$ lat  :List of 3
 .. .. ..$ Type  : chr "Lat"
 .. .. ..$ Units : chr "degrees north"
 .. .. ..$ Values: num [1:6] 35 37.5 40 42.5 45 47.5
 .. ..$ lon  :List of 3
 .. .. ..$ Type  : chr "Lon"
 .. .. ..$ Units : chr "degrees east"
 .. .. ..$ Values: num [1:9] -15 -12.5 -10 -7.5 -5 -2.5 0 2.5 5
$ SLPd:List of 5
 ..$ Description: chr "Mean Sea Level Pressure; Mean daily value"
 ..$ DataType   : chr "float"
 ..$ Units      : chr "Pa"
 ..$ TimeStep   :Class 'difftime'  atomic [1:1] 24
 .. .. ..- attr(*, "tzone")= chr ""
 .. .. ..- attr(*, "units")= chr "hours"
 ..$ Dimensions :List of 3
 .. ..$ time:List of 3
 .. .. ..$ Type  : chr "Time"
 .. .. ..$ Units : chr "days since 1950-01-01 00:00:00"
 .. .. ..$ Values: POSIXlt[1:16071], format: "1958-01-01" "1958-01-02" "1958-01-03" "1958-01-04" ...
 .. ..$ lat :List of 3
 .. .. ..$ Type  : chr "Lat"
 .. .. ..$ Units : chr "degrees north"
 .. .. ..$ Values: num [1:6] 35 37.5 40 42.5 45 47.5
 .. ..$ lon :List of 3
 .. .. ..$ Type  : chr "Lon"
 .. .. ..$ Units : chr "degrees east"
 .. .. ..$ Values: num [1:9] -15 -12.5 -10 -7.5 -5 -2.5 0 2.5 5
$ T   :List of 5
 ..$ Description: chr "Temperature"
 ..$ DataType   : chr "float"
 ..$ Units      : chr "K"
 ..$ TimeStep   :Class 'difftime'  atomic [1:1] 24
 .. .. ..- attr(*, "tzone")= chr ""
 .. .. ..- attr(*, "units")= chr "hours"
 ..$ Dimensions :List of 4
 .. ..$ level:List of 3
 .. .. ..$ Type  : chr "Pressure"
 .. .. ..$ Units : chr "millibar"
 .. .. ..$ Values: num 850
 .. ..$ time :List of 3
 .. .. ..$ Type  : chr "Time"
 .. .. ..$ Units : chr "days since 1950-01-01 00:00:00"
 .. .. ..$ Values: POSIXlt[1:16071], format: "1958-01-01" "1958-01-02" "1958-01-03" "1958-01-04" ...
 .. ..$ lat  :List of 3
```

```
 .. .. ..$ Type  : chr "Lat"
 .. .. ..$ Units : chr "degrees north"
 .. .. ..$ Values: num [1:6] 35 37.5 40 42.5 45 47.5
 .. ..$ lon  :List of 3
 .. .. ..$ Type  : chr "Lon"
 .. .. ..$ Units : chr "degrees east"
 .. .. ..$ Values: num [1:9] -15 -12.5 -10 -7.5 -5 -2.5 0 2.5 5
$ Z   :List of 5
 ..$ Description: chr "Geopotential"
 ..$ DataType   : chr "float"
 ..$ Units      : chr "m**2 s**-2"
 ..$ TimeStep   :Class 'difftime'  atomic [1:1] 24
 .. .. ..- attr(*, "tzone")= chr ""
 .. .. ..- attr(*, "units")= chr "hours"
 ..$ Dimensions :List of 4
 .. ..$ level:List of 3
 .. .. ..$ Type  : chr "Pressure"
 .. .. ..$ Units : chr "millibar"
 .. .. ..$ Values: num 850
 .. ..$ time :List of 3
 .. .. ..$ Type  : chr "Time"
 .. .. ..$ Units : chr "days since 1950-01-01 00:00:00"
 .. .. ..$ Values: POSIXlt[1:16071], format: "1958-01-01" "1958-01-02" "1958-01-03" "1958-01-04" ...
 .. ..$ lat  :List of 3
 .. .. ..$ Type  : chr "Lat"
 .. .. ..$ Units : chr "degrees north"
 .. .. ..$ Values: num [1:6] 35 37.5 40 42.5 45 47.5
 .. ..$ lon  :List of 3
 .. .. ..$ Type  : chr "Lon"
 .. .. ..$ Units : chr "degrees east"
 .. .. ..$ Values: num [1:9] -15 -12.5 -10 -7.5 -5 -2.5 0 2.5 5
```

As we can see, the inventory consists of a list of four elements, which are the four variables stored in the dataset:

```
> names(inv.iberiaNCEP)
[1] "Q"    "SLPd" "T"    "Z"
```

This is how we can check the spatial domain of the dataset:

```
> lats = inv.iberiaNCEP$SLPd$Dimensions$lat$Values
> lons = inv.iberiaNCEP$SLPd$Dimensions$lon$Values
> plot(expand.grid(lons,lats), asp=1)
# 'world' from library 'fields'
> world(add=TRUE)
```

loadGCM

Once the NcML dataset is created and we get an idea of the nature of the variables stored, the `loadGCM` function is used to retrieve the variables desired at selected dimensional slices. Although the name of the function may result somewhat misleading, the function is intended for loading many kinds of gridded datasets, and not only GCM data, including reanalysis, RCM data and observational gridded datasets, for instance. In this particular example, we will load the temperature data from the NCEP reanalysis in the Iberian Peninsula, provided in the example datasets of the meteoR package.

We have a look again to the description of the variable temperature, as provided by the `dataInventory`:

```
> str(inv.iberiaNCEP$T)
List of 5
$ Description: chr "Temperature"
$ DataType   : chr "float"
$ Units      : chr "K"
$ TimeStep   :Class 'difftime'  atomic [1:1] 24
 .. ..- attr(*, "tzone")= chr ""
 .. ..- attr(*, "units")= chr "hours"
$ Dimensions :List of 4
 ..$ level:List of 3
 .. ..$ Type  : chr "Pressure"
 .. ..$ Units : chr "millibar"
 .. ..$ Values: num 850
 ..$ time :List of 3
 .. ..$ Type  : chr "Time"
 .. ..$ Units : chr "days since 1950-01-01 00:00:00"
 .. ..$ Values: POSIXlt[1:16071], format: "1958-01-01" "1958-01-02" "1958-01-03" "1958-01-04" ...
 ..$ lat  :List of 3
 .. ..$ Type  : chr "Lat"
 .. ..$ Units : chr "degrees north"
 .. ..$ Values: num [1:6] 35 37.5 40 42.5 45 47.5
 ..$ lon  :List of 3
 .. ..$ Type  : chr "Lon"
 .. ..$ Units : chr "degrees east"
 .. ..$ Values: num [1:9] -15 -12.5 -10 -7.5 -5 -2.5 0 2.5 5
```

As we can see, the variable T has vertical levels. In this case, the only level available is at 850 mb. The variable is daily, as we can see in the `TimeStep` element of the list, and the original units are Kelvin.

There are several options for spatial selection using the `loadGCM` function, as in the case of `loadSeasonalForecast`. For instance, if we want the whole domain of the dataset, there is no need for specifying the `lonLim` and `latLim` arguments. Alternatively, it is possible to select smaller rectangular domains or single points. In the next example, we will load T850 for January in a similar domain than previously with the SYstem4 dataset, centered on the Iberian Peninsula, encompassing the period 1990-1999.

```
> t850.ncep.iberia <- loadGCM(dataset = "./datasets/reanalysis/Iberia_NCEP/Iberia_NCEP.ncml", standard.vars=TRUE,
+        var="ta", lonLim = c(-10,5), latLim = c(35,45), level=850, season=1, years=1990:1999)
> str(t850.ncep.iberia)
List of 5
$ VarName     : chr "ta"
$ Level       : num 850
$ Dates       :List of 2
 ..$ Start: POSIXlt[1:310], format: "1990-01-01" "1990-01-02" "1990-01-03" "1990-01-04" ...
 ..$ End  : POSIXlt[1:310], format: "1990-01-01" "1990-01-02" "1990-01-03" "1990-01-04" ...
$ LatLonCoords: num [1:35, 1:2] 35 37.5 40 42.5 45 35 37.5 40 42.5 45 ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : NULL
 .. ..$ : chr [1:2] "lat" "lon"
$ Data        : num [1:310, 1:35] 5.95 5.55 0.35 2.05 5.35 ...
```

Note that in this particular case, we are loading standard variables, as defined in the vocabulary (by setting the argument `standard.vars = TRUE`), but we did not specify a path to the dictionary (default to `NULL`). By default, the function searches the dictionary in the same directory where the *ncml* file has been created, assuming that this is a file with extension *.dic* and the same name as the *ncml*.

The function `loadGCM` allows the retrieval of rectangular windows as well as single point locations. In the last case, the function finds the closest grid point to the given coordinates, as previously described in the example of `loadSeasonalForecast`. In the following lines we extract the time series of the Spanish cities used in the previous examples:

```
> t850.ncep.santander <- loadGCM(dataset = "./datasets/reanalysis/Iberia_NCEP/Iberia_NCEP.ncml", standard.vars=TRUE,
+                            var="ta", lonLim = -3.81, latLim = 43.43, level=850, season=1, years=1990:1999)
```

```
> t850.ncep.madrid <- loadGCM(dataset = "./datasets/reanalysis/Iberia_NCEP/Iberia_NCEP.ncml", standard.vars=TRUE,
+                             var="ta", lonLim = -3.68, latLim = 40.40, level=850, season=1, years=1990:1999)
>
> ylimits <- c(floor(min(t850.ncep.madrid$Data)), ceiling(max(t850.ncep.santander$Data)))
> plot(t850.ncep.santander$Data, ty='n', axes=FALSE, ylab="degC", xlab="Year", ylim = ylimits)
> axis(1,at = seq(1,31*11,31), labels=c(1990:1999,""))
> axis(2, ylim=ylimits)
> abline(v=seq(1,31*10,31), lty=2)
> lines(t850.ncep.santander$Data, ty='l')
> lines(t850.ncep.madrid$Data, col = "red")
> legend("bottomleft", city.names, lty=1, col=1:4)
> title(main = "Mean Temperature January 850mb")
> mtext("NCEP reanalysis")
```