Contents

1 Preface

Using `meteoland` package is easy, but some ideas and concepts must be addresed to make the experience easy for new users. This manual is intended as a working example to explain all the steps needed to get comfortable with `meteoland` workflow.

2 Installation and load

First of all, before starting to work with the package, we must install and load the library. `meteoland` stable version can be found at CRAN (https://cran.r-project.org/web/packages/meteoland/index.html), and it can be installed and loaded as any other R package:

```
install.packages("meteoland")
library(meteoland)
```

2.1 Development version

You can install the development version located at github using the `devtools` package

```
library(devtools)
install_github('miquelcaceres/meteoland')
library(meteoland)
```

3 Shaping the data

Before starting with interpolations/downscaling/corrections, we need to get the stations data in shape. But be warned, depending on the format stations data come, you will have to perform different steps than the ones we will describe here, as there is no universal format for metereological stations data.

3.1 Required data format

3.1.1 Stations data format

In order to create an interpolation object to work with, we need a matrix for each variable with the days as columns and the weather stations as rows, as we can see in these minimum temperature recordings:

```
head(tmin[,1:6])
```

```
##       1976-01-01 1976-01-02 1976-01-03 1976-01-04 1976-01-05 1976-01-06
## ST_01 -3.0320040  -3.586816  -2.295752 -2.1678246 -2.1865915 -2.2797344
## ST_02 -3.8826743  -4.162509  -1.104466  0.3146304 -0.9274811 -1.0097591
## ST_03 -2.7094458  -2.992890  -1.407178 -0.6581457 -0.9338085 -0.8136145
## ST_04 -2.0967963  -3.073414  -1.169255  0.8583836 -1.1333813 -2.1484118
## ST_05 -1.5024432  -2.226421  -1.476496 -0.1027094 -0.8091650 -1.0908050
## ST_06 -0.9151786  -1.949163  -1.370876  0.5939252 -1.0959637 -2.1859451
```

3.1.2 Stations coordinates

Also we need a data frame with the stations coordinates:

```
head(stations_info)
```

```
## # A tibble: 6 x 4
##    Code      Name Latitude Longitude
##   <chr>     <chr>    <dbl>     <dbl>
## 1 ST_01 Station 1    42.04      1.37
## 2 ST_02 Station 2    42.04      1.57
## 3 ST_03 Station 3    41.92      1.45
## 4 ST_04 Station 4    41.91      1.60
## 5 ST_05 Station 5    41.85      1.50
## 6 ST_06 Station 6    41.98      1.32
```

3.1.3 Stations topography

Finally, we are gonna need the stations topography info (elevation, aspect and slope):

```
head(stations_topo)
```

```
## # A tibble: 6 x 4
##    Code Elevation Slope Aspect
##   <chr>     <dbl> <dbl>  <dbl>
## 1 ST_01       635  6.60 149.74
## 2 ST_02       690 15.18 259.38
## 3 ST_03       796  4.69 293.96
## 4 ST_04       677  5.01 244.65
## 5 ST_05       687  2.64 275.19
## 6 ST_06       503  8.54 273.18
```

Both stations coordinates and stations topography can be in the same data frame, but usually you will obtain the topography from different sources than the meteorological data, so in this example they will be separate.

3.2 Getting the meteorological data in shape

Now that we know how the data must be formatted, here we will describe an user case on how to transform the data to the preferred format.
Usually, data from weather stations comes in one file for each station, containing all the variables measured by that station, for example:

```
st_01 <- read.table('ST_01.txt', header = TRUE, sep = '\t', dec = '.',
                    row.names = 1)

head(st_01)
```

```
##            MaxTemperature MinTemperature Radiation Precipitation
## 1976-01-01       6.368068      -3.032004  7.575003             0
## 1976-01-02       8.199486      -3.586816  8.491354             0
## 1976-01-03       8.868364      -2.295752  8.341635             0
## 1976-01-04       7.715426      -2.167825  7.916156             0
## 1976-01-05      11.066443      -2.186591  8.917625             0
## 1976-01-06       8.051418      -2.279734  8.187080             0
##            RelativeHumidity WindSpeed
## 1976-01-01        100.00000  2.037037
## 1976-01-02         86.40331  1.944444
## 1976-01-03         82.72656  3.333333
## 1976-01-04         92.80110  4.351852
## 1976-01-05         82.73723  2.685185
## 1976-01-06         96.49268  4.074074
```

Here we found variables as columns and dates as rows. So, based in the format required to work with `meteoland`, we need to read all the stations files and create an object for each variable, but with dates as columns and stations as rows. Lets do that for 10 different stations.

First we put the stations and the dates we want to retrieve in two vectors and create the empty objects where we are going to store the data:

```
# stations and dates
stations <- c('ST_01', 'ST_02', 'ST_03', 'ST_04', 'ST_05',
              'ST_06', 'ST_07', 'ST_08', 'ST_09', 'ST_10')
dates <- as.character(
  seq(as.Date('1976-01-01'), as.Date('2016-12-31'), by = 'day')
)

# empty variable objects
tmin <- data.frame(
  matrix(
    NA,
    ncol = length(dates),
    nrow = length(stations)
  )
)

row.names(tmin) <- stations
names(tmin) <- dates

tmax <- tmin
rh <- tmin
prec <- tmin
```

```
sr <- tmin
ws <- tmin
wd <- tmin
```

Now we have all the variables, with stations as rows and dates as columns, as we want, but they are empty, so we need to fill them with the data. To do that we are gonna use a loop, which will read each station file and retrieve the weather variables data:

```
for (station in stations) {
  file_name <- paste0(station, '.txt')

  # check if file exists to avoid errors
  if (file.exists(file_name)) {
    station_data <- read.table(file_name, header = TRUE, sep = '\t',
                               dec = '.', row.names = 1)
    station_dates <- row.names(station_data)

    # check for variables
    if ('MinTemperature' %in% names(station_data)) {
      tmin[station, dates] <- station_data$MinTemperature
    }

    if ('MaxTemperature' %in% names(station_data)) {
      tmax[station, dates] <- station_data$MaxTemperature
    }

    if ('Precipitation' %in% names(station_data)) {
      prec[station, dates] <- station_data$Precipitation
    }

    if ('RelativeHumidity' %in% names(station_data)) {
      rh[station, dates] <- station_data$RelativeHumidity
    }

    if ('Radiation' %in% names(station_data)) {
      sr[station, dates] <- station_data$Radiation
    }

    if ('WindSpeed' %in% names(station_data)) {
      ws[station, dates] <- station_data$WindSpeed
    }

    if ('WindDirection' %in% names(station_data)) {
      wd[station, dates] <- station_data$WindDirection
    }
  }
}
```

Sometimes, we may also need to check the quality of the data retrieved from stations. For example, being this example data from a mediterranean area, we suspect that temperature values below -30 or above 50 Celsius degrees can be due to measure errors in the weather stations. So we can make a custom function to remove these dubious values:

```
checkTemperature <- function(temp) {
  temp[(!is.na(temp)) & (temp < (-30))] = NA
```

```r
  temp[(!is.na(temp)) & (temp > 50)] = NA
  return(temp)
}

tmin <- checkTemperature(tmin)
tmax <- checkTemperature(tmax)
```

Also, we can do this for any other variable, we only need to write the custom function we want and clean the data.

3.3 Building the interpolation object

Now that we have the variables data in format, as well as the topographical info and the coordinates of the stations we are ready to build the interpolation object we are gonna use with `meteoland` package. If we look at the `MeteorologyInterpolationData` function help (`?MeteorologyInterpolationData`) we can see that we need the coordinates in a `SpatialPoints` (from the `sp` package) class object. This will allow to establish the projection of the coordinates[1]:

```r
# sp library
library(sp)
# spatial points object
coords_example <- SpatialPoints(
  cbind(stations_info$Longitude, stations_info$Latitude),
  proj4string = CRS('+init=epsg:4326')
)
```

And finally, we can build the interpolation object:

```r
interpolator <- MeteorologyInterpolationData(
  coords_example,
  elevation = stations_topo$Elevation,
  slope = stations_topo$Slope,
  aspect = stations_topo$Aspect,
  MinTemperature = as.matrix(tmin),
  MaxTemperature = as.matrix(tmax),
  Precipitation = as.matrix(prec),
  RelativeHumidity = as.matrix(rh),
  Radiation = as.matrix(sr),
  WindSpeed = as.matrix(ws),
  WindDirection = as.matrix(wd),
  params = defaultInterpolationParams()
)
```

4 Calibration and Validation

Once we already have the meteorlogical stations data in shape, we can start calibrating the model in order to obtain the optimal parameters for the meteorological variables we want to interpolate.

[1] In this case is standard longitude/latitude coordinates in the *mercator* projection

4.1 Stations data coverage

Sometimes can be useful to summarise the temporal and spatial coverage of our data:

```
spatial_coverage <- interpolation.coverage(interpolator, type = 'spatial')
head(spatial_coverage)
```

```
##         coordinates MinTemperature MaxTemperature Precipitation
## ST_01 (1.37, 42.04)          14976          14976         14976
## ST_02 (1.57, 42.04)          14976          14976         14976
## ST_03 (1.45, 41.92)          14976          14976         14976
## ST_04  (1.6, 41.91)          14976          14976         14976
## ST_05  (1.5, 41.85)          14976          14976         14976
## ST_06 (1.32, 41.98)          14976          14976         14976
##       RelativeHumidity Radiation WindSpeed WindDirection
## ST_01            14976     14976     14180             0
## ST_02            14976     14976     13908             0
## ST_03            14976     14976     13898             0
## ST_04            14976     14976     11096             0
## ST_05            14976     14976     12571             0
## ST_06            14976     14976     11304             0
```

```
temporal_coverage <- interpolation.coverage(interpolator, type = 'temporal')
head(temporal_coverage)
```

```
##            MinTemperature MaxTemperature Precipitation RelativeHumidity
## 1976-01-01             10             10            10               10
## 1976-01-02             10             10            10               10
## 1976-01-03             10             10            10               10
## 1976-01-04             10             10            10               10
## 1976-01-05             10             10            10               10
## 1976-01-06             10             10            10               10
##            Radiation WindSpeed WindDirection
## 1976-01-01        10         5             0
## 1976-01-02        10         5             0
## 1976-01-03        10         6             0
## 1976-01-04        10         6             0
## 1976-01-05        10         6             0
## 1976-01-06        10         6             0
```

As you can see, `interpolation.coverage` function is used to summarise the coverage of our data. Specifing `type = 'spatial'` returns an `SpatialPointsDataFrame` class object with the number of dates with data per station and metereological variable, whereas `type = 'temporal'` returns a data frame with the number of stations with data per day (rows) and meterological variable (columns).

It seems like the data is ok, so we can go further to the next step.

4.2 Calibration

Calibration must be done for each desired variable, being the process the same for each variable. We will use `interpolation.calibration` function, which need to be supplied with the stations data (a `MetereologyInterpolationData` class object), the variable name as a string and finally, N and α values to be tested:

```
tmin_cal <- interpolation.calibration(interpolator, variable = "Tmin",
                                      N_seq = seq(5, 10, by = 1),
```

```r
                                           alpha_seq = seq(0.5, 10, by = 0.5),
                                           verbose = TRUE)

tmax_cal <- interpolation.calibration(interpolator, variable = "Tmax",
                                           N_seq = seq(5, 10, by = 1),
                                           alpha_seq = seq(0.5, 10, by = 0.5),
                                           verbose = TRUE)

tdew_cal <- interpolation.calibration(interpolator, variable = "Tdew",
                                           N_seq = seq(5, 10, by = 1),
                                           alpha_seq = seq(0.5, 10, by = 0.5),
                                           verbose = TRUE)

prec_cal <- interpolation.calibration(interpolator, variable = "Prec",
                                           N_seq = seq(5, 10, by = 1),
                                           alpha_seq = seq(0.5, 10, by = 0.5),
                                           verbose = TRUE)
```

This function returns an `interpolation.calibration` class object which contains several items:

- Numeric matrix with the mean absolute error (MAE) values for each combination of parameters N and α.
- Miminum value found for MAE.
- Value for the N parameter corresponding to the minumun MAE.
- Value for the α parameter corresponding to the minimum MAE.
- Matrix with the observed values.
- Matrix with the predicted values for the optimum parameter combination.

We can see the MAE and parameter values found in the calibration:

```r
tmin_cal$minMAE
## [1] 2.028257
tmin_cal$N
## [1] 5
tmin_cal$alpha
## [1] 0.5
```

We can also see the results visually, with `lattice`:

(ref:tmin_cal_plot) Contour plot for minimum temperature calibration.

```r
z = tmin_cal$MAE
x = as.numeric(rownames(tmin_cal$MAE))
y = as.numeric(colnames(tmin_cal$MAE))
filled.contour(x,y,z, main = "Minimum Temperature",
               plot.axes = {
                 points(tmin_cal$N, tmin_cal$alpha, cex = 1, pch = 4)
                 axis(1, 1:15)
                 axis(2, 1:15)
               },
               xlab = "Station density (N)",
               ylab = "Shape parameter (alpha)",
               color.palette = colorRampPalette(c("white","red","black")))
```

(ref:tmax_cal_plot) Contour plot for maximum temperature calibration.

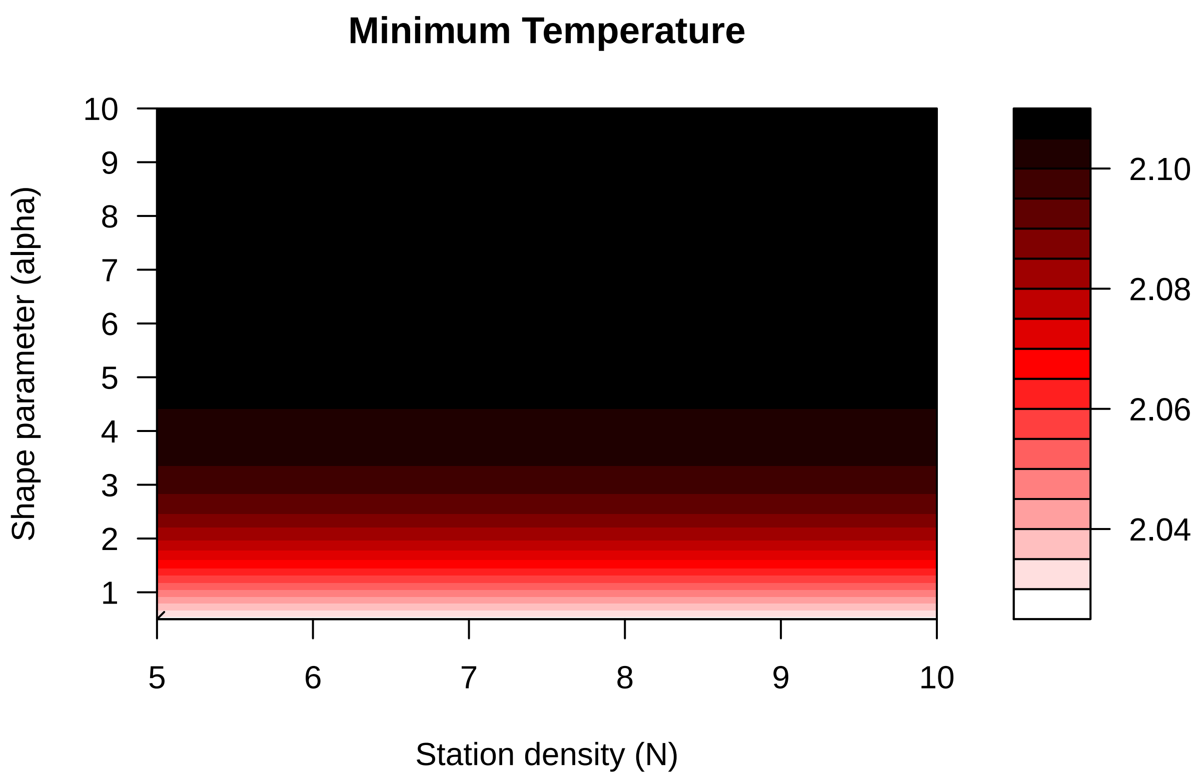


Figure 1: (#fig:tmin_cal_plot)(ref:tmin_cal_plot)

```
z = tmax_cal$MAE
x = as.numeric(rownames(tmax_cal$MAE))
y = as.numeric(colnames(tmax_cal$MAE))
filled.contour(x,y,z, main = "Maximum Temperature",
               plot.axes = {
                 points(tmax_cal$N, tmax_cal$alpha, cex = 1, pch = 4)
                 axis(1, 1:15)
                 axis(2, 1:15)
               },
               xlab = "Station density (N)",
               ylab = "Shape parameter (alpha)",
               color.palette = colorRampPalette(c("white","red","black")))
```

(ref:tdew_cal_plot) Contour plot for dew temperature calibration.

```
z = tdew_cal$MAE
x = as.numeric(rownames(tdew_cal$MAE))
y = as.numeric(colnames(tdew_cal$MAE))
filled.contour(x,y,z, main = "Dew Temperature",
               plot.axes = {
                 points(tdew_cal$N, tdew_cal$alpha, cex = 1, pch = 4)
                 axis(1, 1:15)
                 axis(2, 1:15)
               },
               xlab = "Station density (N)",
               ylab = "Shape parameter (alpha)",
```

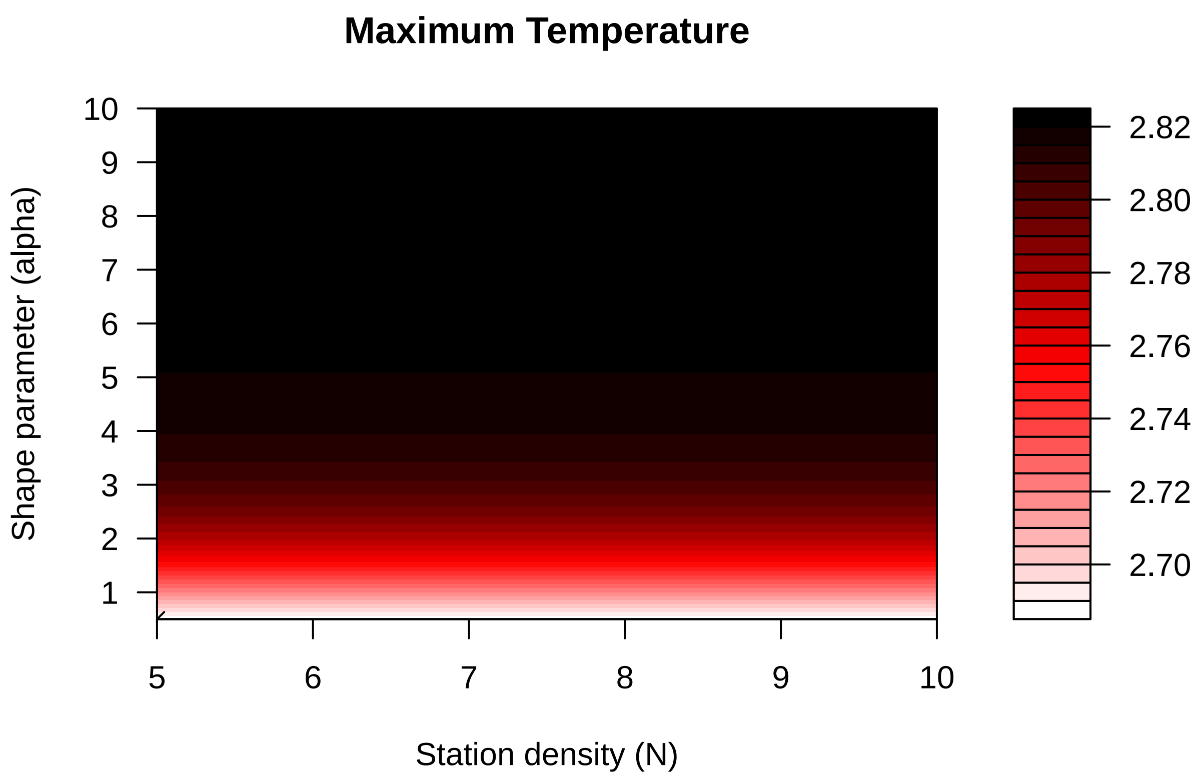


Figure 2: (#fig:tmax_cal_plot)(ref:tmax_cal_plot)

```
               color.palette = colorRampPalette(c("white","red","black")))
```

(ref:prec_cal_plot) Contour plot for precipitation calibration.

```
z = prec_cal$MAE
x = as.numeric(rownames(prec_cal$MAE))
y = as.numeric(colnames(prec_cal$MAE))
filled.contour(x,y,z, main = "Precipitation",
               plot.axes = {
                 points(prec_cal$N, prec_cal$alpha, cex = 1, pch = 4)
                 axis(1, 1:15)
                 axis(2, 1:15)
               },
               xlab = "Station density (N)",
               ylab = "Shape parameter (alpha)",
               color.palette = colorRampPalette(c("white","red","black")))
```

Seems ok, so we need to store the parameter values for each variable in the interpolator data:

```
interpolator@params = defaultInterpolationParams()
interpolator@params$N_MinTemperature = tmin_cal$N
interpolator@params$alpha_MinTemperature = tmin_cal$alpha
interpolator@params$N_MaxTemperature = tmax_cal$N
interpolator@params$alpha_MaxTemperature = tmax_cal$alpha
interpolator@params$N_DewTemperature = tdew_cal$N
interpolator@params$alpha_DewTemperature = tdew_cal$alpha
interpolator@params$N_PrecipitationEvent = prec_cal$N
```

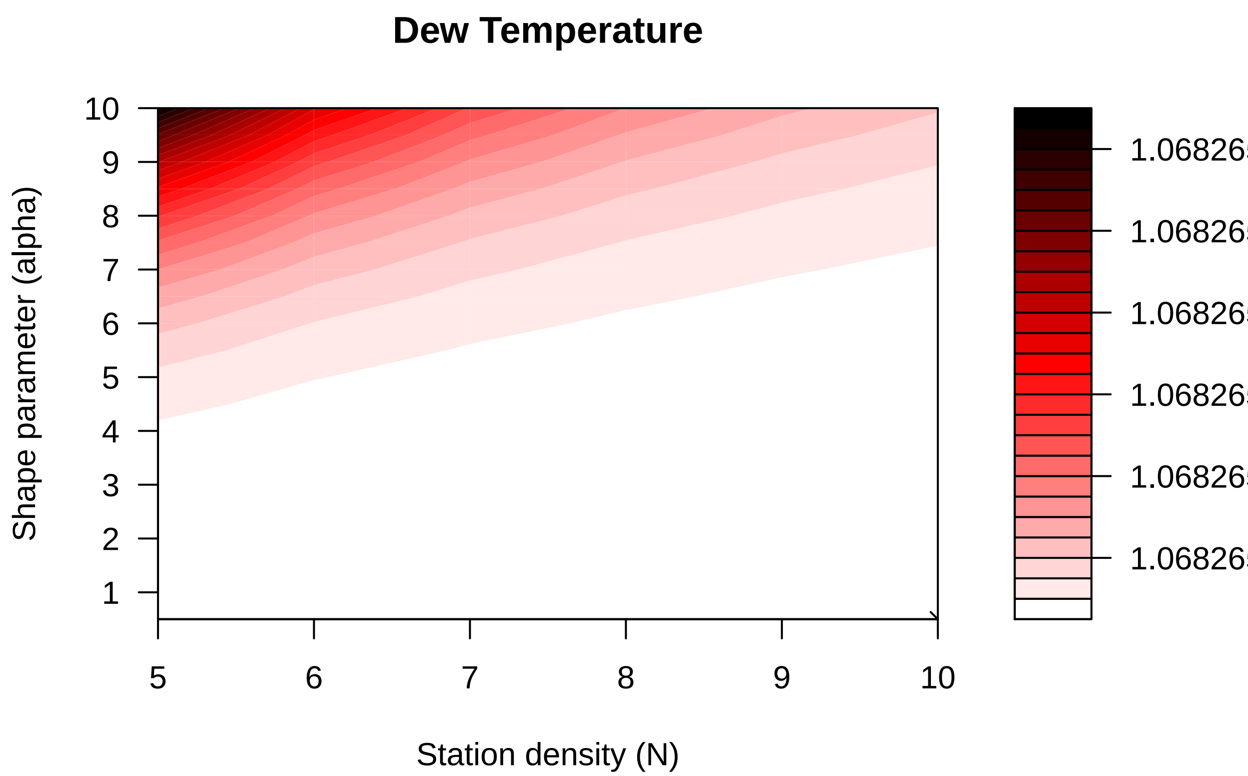


Figure 3: (#fig:tdew_cal_plot)(ref:tdew_cal_plot)

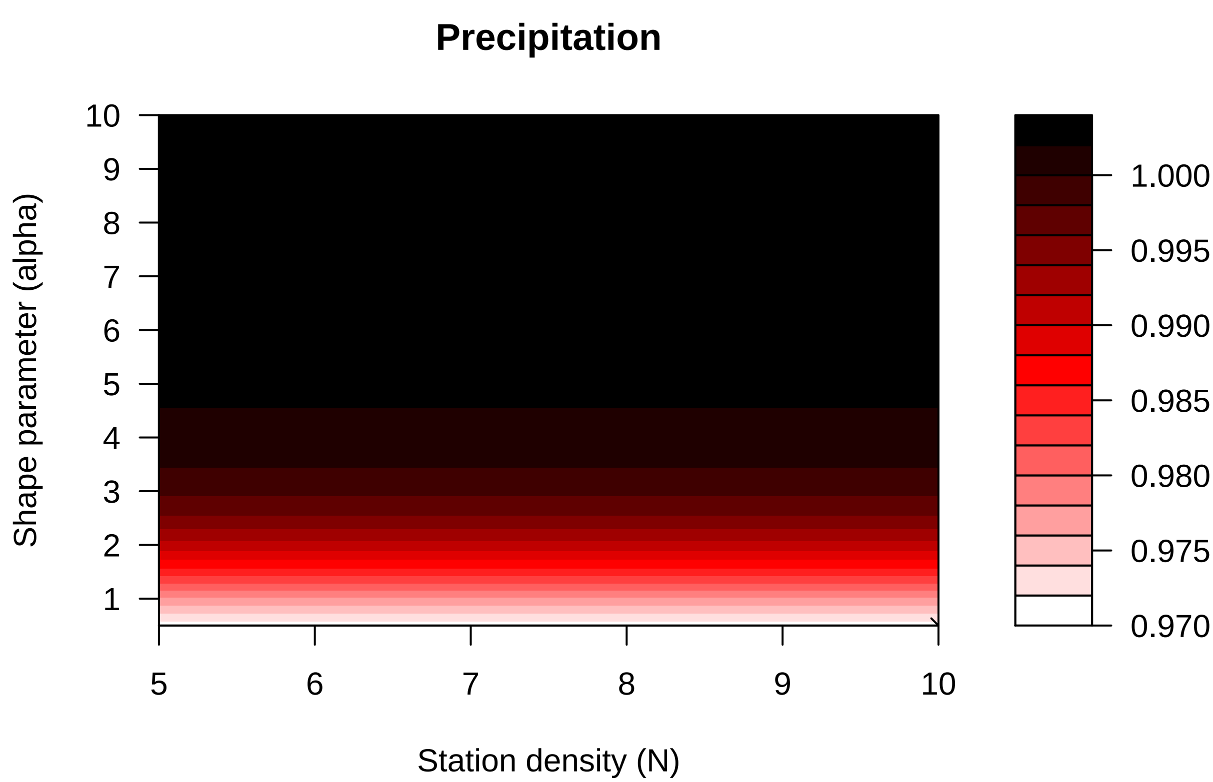


Figure 4: (#fig:prec_cal_plot)(ref:prec_cal_plot)

```r
interpolator@params$alpha_PrecipitationEvent = prec_cal$alpha
interpolator@params$N_PrecipitationAmount = prec_cal$N
interpolator@params$alpha_PrecipitationAmount = prec_cal$alpha
interpolator@params$St_Precipitation = 5
interpolator@params$pop_crit = 0.50
interpolator@params$f_max = 0.95
```

4.3 Cross Validation

In order to perfom the interpolation validation we will use the `interpolation.cv` function:

```r
cv <- interpolation.cv(interpolator)
```

And now we can inspect the results obtained and plot them:

```r
summary(cv)
```

```
##                                   n          r          MAE sd.station.MAE
## MinTemperature               149756  0.8609969     2.025623       2.227480
## MaxTemperature               149759  0.8697528     2.685616       2.983204
## TemperatureRange             149755  0.5709987     2.172816       2.172069
## RelativeHumidity             149760  0.7584507     7.492475       7.004435
## Radiation                    149760  0.9038170     2.234068       1.503315
## Station.rainfall                 10 -0.1413738 14439.443937   26491.247481
## Station.rainfall.relative        10         NA    56.982108     109.267906
## Station.precdays                 10 -0.8143509   718.600000     311.306280
## Station.precdays.relative        10         NA    18.488228       5.998209
## Date.rainfall                  3704  0.7059248    33.032421             NA
## Date.rainfall.relative         3704         NA    61.751332             NA
## Date.precstations              3704  0.6944854     1.325864             NA
## Date.precstations.relative     3704         NA    19.197500             NA
##                              sd.dates.MAE          Bias sd.station.Bias
## MinTemperature                  1.3043778    0.64210780    2.739458e+00
## MaxTemperature                  1.3970109    0.58622621    2.883250e+00
## TemperatureRange                1.5113458   -0.05616653    8.093868e-01
## RelativeHumidity                2.8770380   -2.06174052    8.054737e+00
## Radiation                       0.9779347    0.13584510    1.838516e+00
## Station.rainfall                       NA 9047.40275770    2.902584e+04
## Station.rainfall.relative              NA   43.20875582    1.160723e+02
## Station.precdays                       NA -420.20000000    6.888302e+02
## Station.precdays.relative              NA   -8.47868389    1.832749e+01
## Date.rainfall                 107.7160297   30.93055965              NA
## Date.rainfall.relative        127.7105706   59.00386004              NA
## Date.precstations               1.4694368    1.32586393              NA
## Date.precstations.relative     23.3185520   19.19749991              NA
##                              sd.dates.Bias
## MinTemperature                   0.5727662
## MaxTemperature                   0.8394303
## TemperatureRange                 0.8941010
## RelativeHumidity                 1.5756846
## Radiation                        0.9565330
## Station.rainfall                        NA
## Station.rainfall.relative               NA
## Station.precdays                        NA
```

```
## Station.precdays.relative              NA
## Date.rainfall                   108.3384542
## Date.rainfall.relative          129.0032983
## Date.precstations                 1.4694368
## Date.precstations.relative       23.3185520
```

(ref:cv_plot) Cross validation plots.

```r
plot(cv)
```

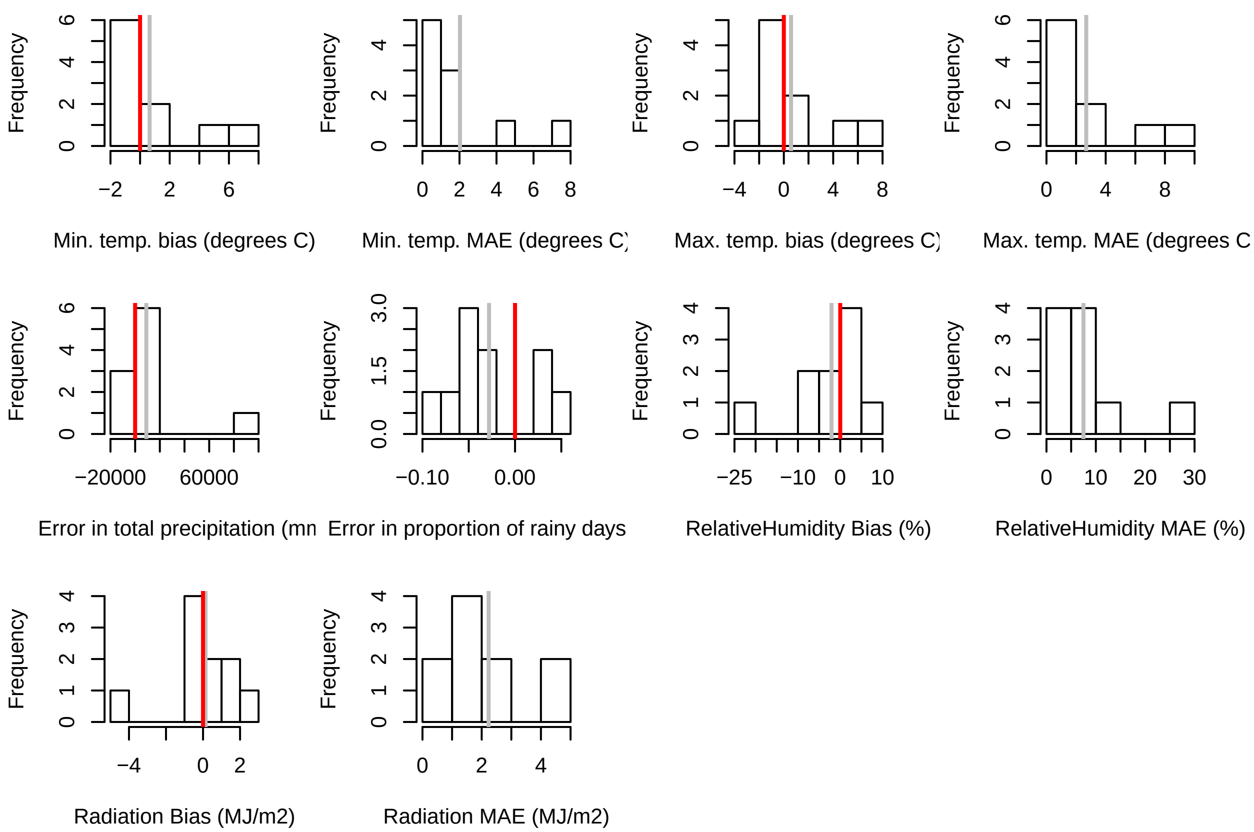


Figure 5: (#fig:plot_cv)(ref:cv_plot)

As we can see, the cross-validation process reassure the calibration parameters obtained, with low *Bias* and *MAE* values for each variable.

Now we are ready to start the interpolation process, what we will see in the next chapter.

5 Interpolation

In this chapter we will see how to interpolate historical data to the landscape scale using the meteorological stations data we build in the previous chapters.

5.1 Points interpolation

If we take a look to the `interpolationpoints` function help (`?interpolationpoints`) we can see what we need to perform the interpolation:

- A `MetereologyInterpolationData` class object, which we obtained from the previous chapters.
- A vector of `Date` class with the dates when we want the interpolation.
- A `SpatialPointsTopography` object with the coordinates and the topographic information of the points we want to interpolate.

We have the first item in the list, and the second one is as easy as `seq(as.Date(first_day), as.Date(last_day), by = 'day')` where `first_day` and `last_day` are the first and last day of the period we want, respectively. In our case we want to interpolate all 2010 period, so we build the dates object:

```
dates_interpolation <- seq(as.Date('2010-01-01'), as.Date('2010-12-31'), by = 'day')
```

Regarding to the third element, the `SpatialPointsTopography` object, we need to build it from the information about the points we want to interpolate. In this case we want to interpolate data for two experimental plots:

```
# plots coordinates
points_lat <- c(41.79, 42.5)
points_long <- c(1.46, 1.52)
plots_sp <- SpatialPoints(cbind(points_lat, points_long),
                          proj4string = CRS('+init=epsg:4326'))

# plots topography
plots_elevation  <- c(1200, 320)
plots_aspect <- c(253, 45)
plots_slope <- c(12, 30)

# Spatial topography object
plots_topo <- SpatialPointsTopography(plots_sp, plots_elevation, plots_slope, plots_aspect)
```

Now we are ready to perform the interpolation:

```
plots_weather_2010 <- interpolationpoints(
  interpolator, plots_topo, dates_interpolation
)
```

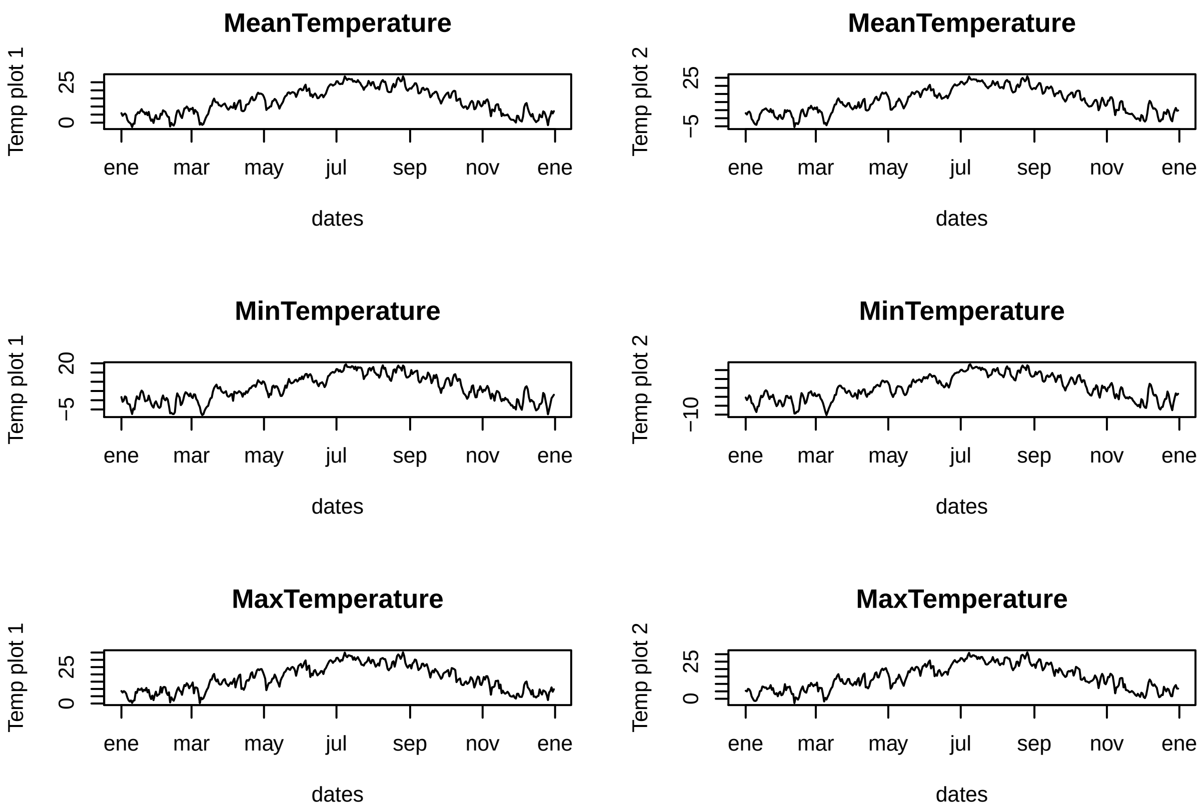
```
## Processing point '1' (1/2) -

## Warning in interpolationpoints(interpolator, plots_topo,
## dates_interpolation): Point '1' outside the boundary box of interpolation
## data object.

##  done.
## Processing point '2' (2/2) -

## Warning in interpolationpoints(interpolator, plots_topo,
## dates_interpolation): Point '2' outside the boundary box of interpolation
## data object.

##  done.
```

In this case, the process is very fast, but things cab become slower as you interpolate longer periods or more points. Anyway, `meteoland` routines are inplemented in `C++`, which make them faster than base `R`.

We can plot the results with the `meteoplot` function:

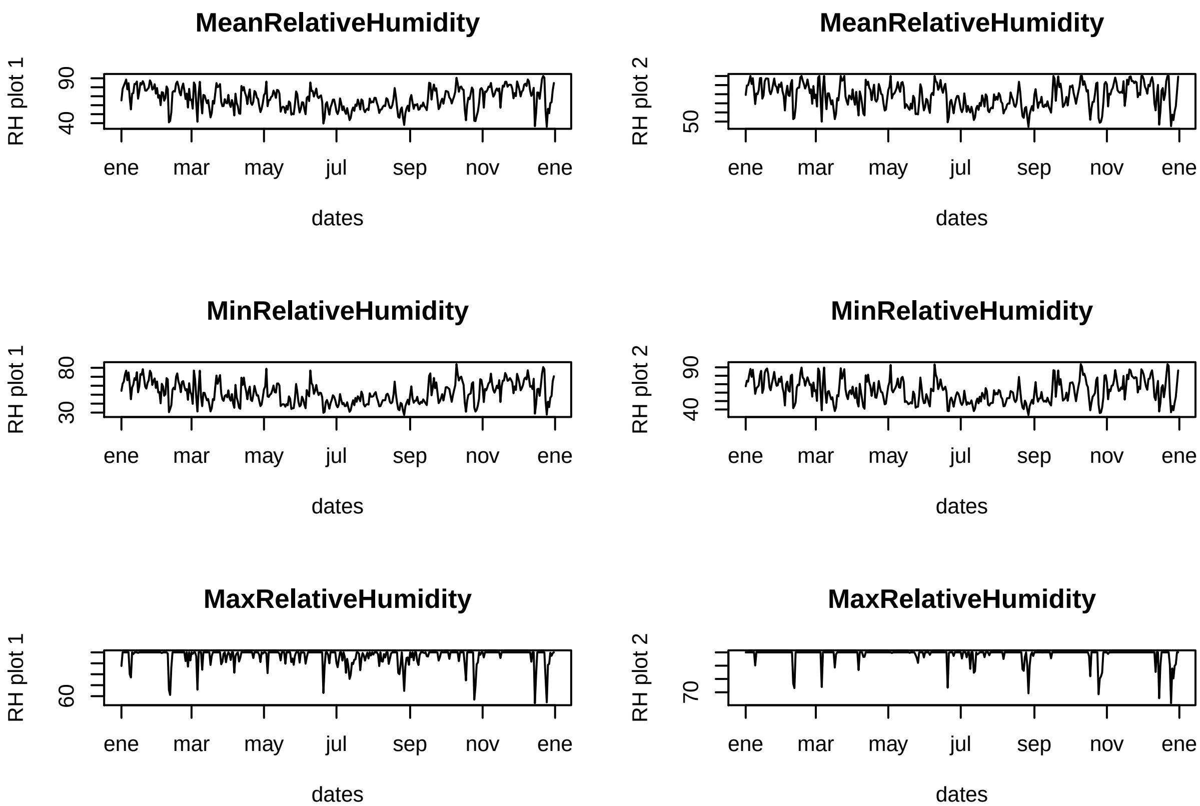
```
par(mfrow = c(3,2))

meteoplot(plots_weather_2010, 1, 'MeanTemperature',
```

```r
          ylab = 'Temp plot 1', main = 'MeanTemperature')
meteoplot(plots_weather_2010, 2, 'MeanTemperature',
          ylab = 'Temp plot 2', main = 'MeanTemperature')
meteoplot(plots_weather_2010, 1, 'MinTemperature',
          ylab = 'Temp plot 1', main = 'MinTemperature')
meteoplot(plots_weather_2010, 2, 'MinTemperature',
          ylab = 'Temp plot 2', main = 'MinTemperature')
meteoplot(plots_weather_2010, 1, 'MaxTemperature',
          ylab = 'Temp plot 1', main = 'MaxTemperature')
meteoplot(plots_weather_2010, 2, 'MaxTemperature',
          ylab = 'Temp plot 2', main = 'MaxTemperature')
```



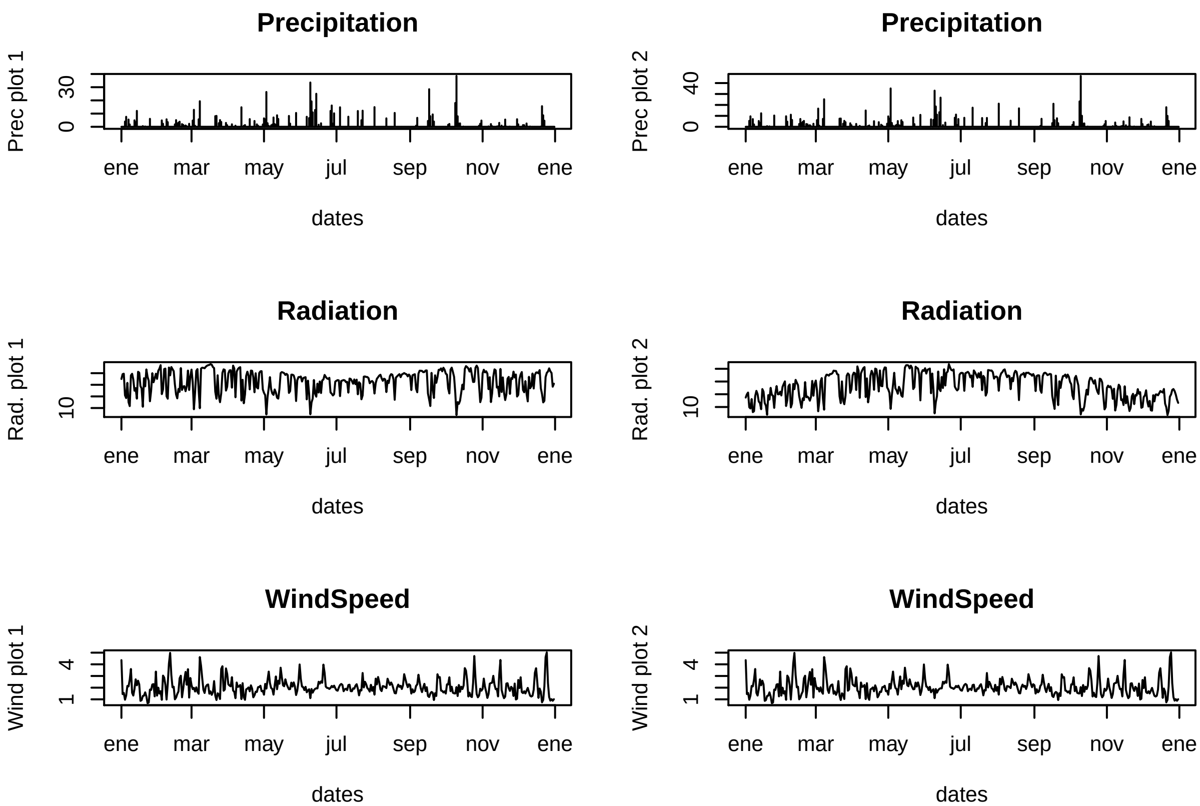

```r
par(mfrow = c(1,1))
```

```r
par(mfrow = c(3,2))

meteoplot(plots_weather_2010, 1, 'MeanRelativeHumidity',
          ylab = 'RH plot 1', main = 'MeanRelativeHumidity')
meteoplot(plots_weather_2010, 2, 'MeanRelativeHumidity',
          ylab = 'RH plot 2', main = 'MeanRelativeHumidity')
meteoplot(plots_weather_2010, 1, 'MinRelativeHumidity',
          ylab = 'RH plot 1', main = 'MinRelativeHumidity')
meteoplot(plots_weather_2010, 2, 'MinRelativeHumidity',
          ylab = 'RH plot 2', main = 'MinRelativeHumidity')
meteoplot(plots_weather_2010, 1, 'MaxRelativeHumidity',
          ylab = 'RH plot 1', main = 'MaxRelativeHumidity')
meteoplot(plots_weather_2010, 2, 'MaxRelativeHumidity',
          ylab = 'RH plot 2', main = 'MaxRelativeHumidity')
```



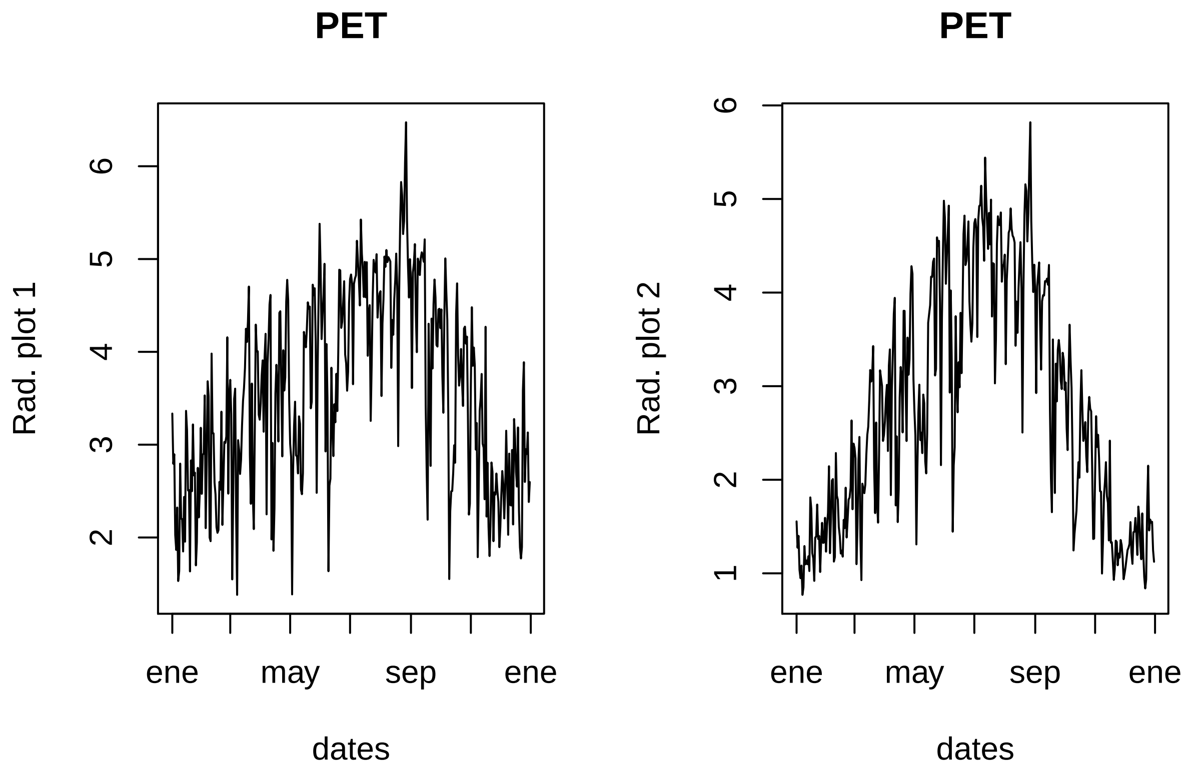

```
par(mfrow = c(1,1))
```

```
par(mfrow = c(3,2))

meteoplot(plots_weather_2010, 1, 'Precipitation',
          ylab = 'Prec plot 1', main = 'Precipitation')
meteoplot(plots_weather_2010, 2, 'Precipitation',
          ylab = 'Prec plot 2', main = 'Precipitation')
meteoplot(plots_weather_2010, 1, 'Radiation',
          ylab = 'Rad. plot 1', main = 'Radiation')
meteoplot(plots_weather_2010, 2, 'Radiation',
          ylab = 'Rad. plot 2', main = 'Radiation')
meteoplot(plots_weather_2010, 1, 'WindSpeed',
          ylab = 'Wind plot 1', main = 'WindSpeed')
meteoplot(plots_weather_2010, 2, 'WindSpeed',
          ylab = 'Wind plot 2', main = 'WindSpeed')
```




```r
par(mfrow = c(1,1))
```

```r
par(mfrow = c(1,2))

meteoplot(plots_weather_2010, 1, 'PET',
          ylab = 'Rad. plot 1', main = 'PET')
meteoplot(plots_weather_2010, 2, 'PET',
          ylab = 'Rad. plot 2', main = 'PET')
meteoplot(plots_weather_2010, 1, 'PET',
          ylab = 'Rad. plot 1', main = 'PET')
meteoplot(plots_weather_2010, 2, 'PET',
          ylab = 'Rad. plot 2', main = 'PET')
```




```
par(mfrow = c(1,1))
```