Third VALUE training workshop, Trieste
Rglimclim practical sessions

Richard E. Chandler
Department of Statistical Science, University College London
r.chandler@ucl.ac.uk

November 5-6, 2014

Contents

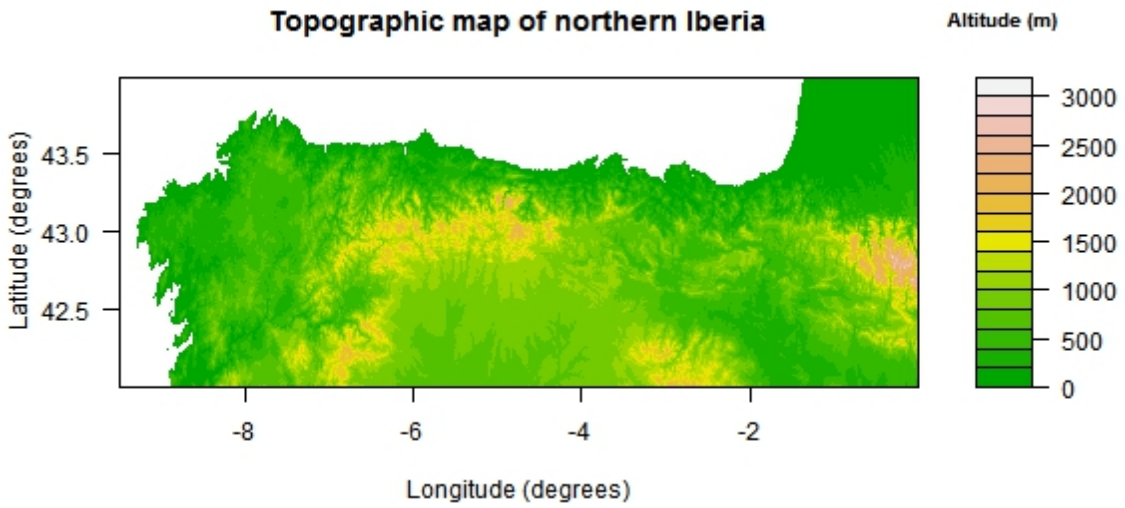

Figure 1: Topographic map of the northern Iberia study area.

1 Introduction

In these practical sessions we will use the Rglimclim package to build and test a bivariate weather generator for daily precipitation and temperature in northern Iberia, defined here as 42°N–44°N, 9.5°W–0°W. A topographic map of the area is shown in Figure 1 — you should recognise this, if you worked through the preparatory materials for the workshop.

This study area is chosen because it is being used to test the VALUE validation framework — see http://www.value-cost.eu/validationTest. The region features dry summers and wet winters, with a climate that is strongly influenced by its proximity to the Atlantic Ocean; and it has enough topographic variation to induce, for example, rain shadow effects on inland south-facing slopes (see http://en.wikipedia.org/wiki/Climate_of_Spain for a summary of the region's climatology).

1.1 Data available

Here is a brief summary of the data that we will use in the practical sessions.

1.1.1 Precipitation and temperature data

The precipitation and temperature data are from the European Climate Assessment & (EC&A) Dataset (**?**), available from http://www.ecad.eu. At the time of writing, the EC&A dataset extends until September 2014 and contains daily precipitation and temperature data from 28 stations within the study area; the earliest record starts in January 1924. We will work with data from 1960 to 2002.[1]

[1]Ideally we would use data for the "official" 1979–2010 VALUE test period. However, I didn't have time to prepare all of the predictor data (see §1.1.3 below) for this period, sorry! — REC.

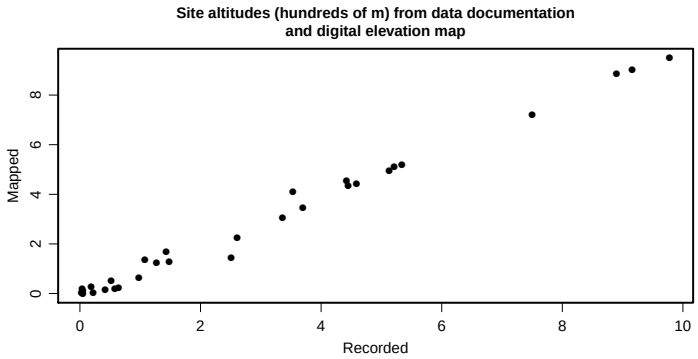

Figure 2: Mapped and exact altitudes for each station in the study area. The "recorded" altitudes are taken from the EC&A station metadata; the "mapped" value for each station is from the GTOPO30 dataset, and is for the 1km^2 grid square containing the station.

The EC&A dataset provides a 'non-blended' data product which contains missing values, and a 'blended' product in which these missing values have been infilled. The blended product is being used elsewhere within VALUE. However, the infilling of missing values can create artificial inhomogeneities in climate records because the infilled values have different statistical properties from the observed values. Moreover, Rglimclim does not require complete data records — it even has its own facilities for stochastic replacement of missing values, so that the user can investigate the associated uncertainties. We will therefore use the 'non-blended' product here.

1.1.2 Topographic data

The EC&A dataset provides geographical coordinates (latitude and longitude) and altitude for each station. We will also use topographic data from the GTOPO30 Digital Elevation Model at http://webmap.ornl.gov/wcsdown/dataset.jsp?ds_id=10003. GTOPO30 provides altitude data for the entire globe at roughly 1km^2 resolution. Figure 1 shows the GTOPO30 data as extracted. These have been used to compute several other topographic indices, as follows:

- Mapped altitude (hundreds of m). This is the value for the 1km^2 GTOPO30 grid square containing each of the EC&A stations. Usually, it differs slightly from the exact altitude of the station. It provides a useful check on the data extraction, however (see Figure 2).
- Mean altitude over squares of dimensions approximately 3×3km^2, 10×10km^2 and 30×30km^2, centred on each station. These will be referred to as '10km^2', '100km^2' and '1000km^2' squares below.
- Standard deviations of altitudes over 10km^2, 100km^2 and 1000km^2 squares. These can be regarded as measures of topographic variability for the different stations, which may have some influence on (for example) the precipitation climatology.

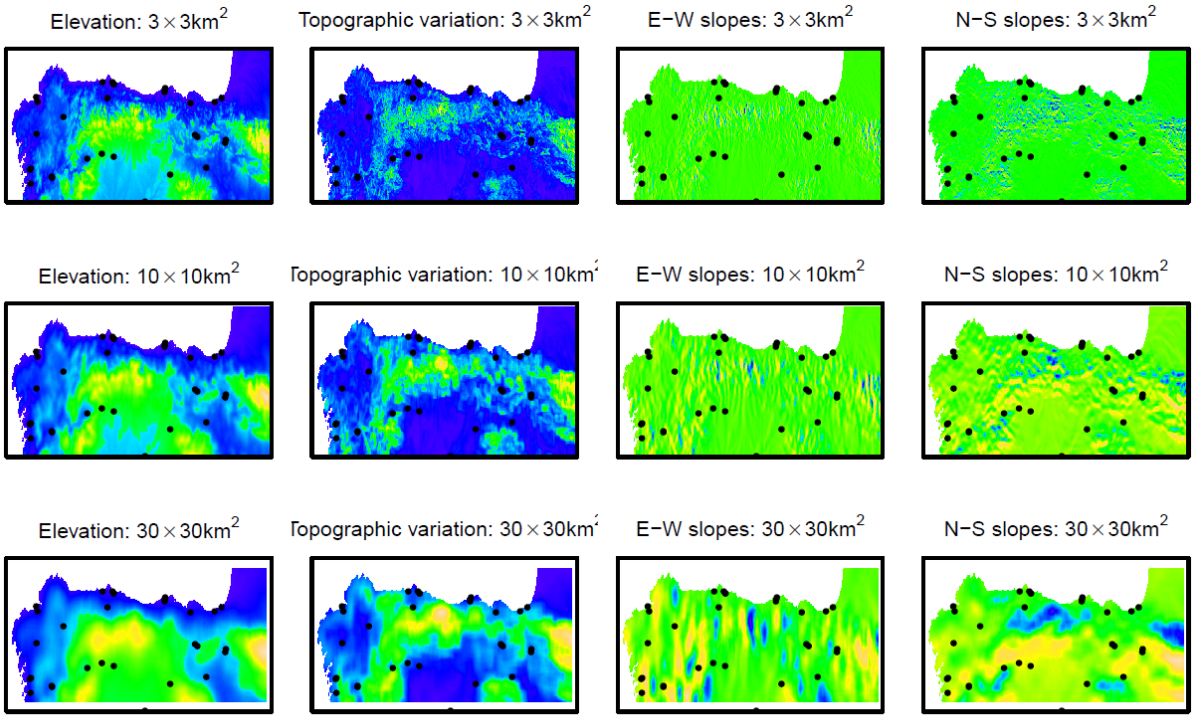

Figure 3: Topographic indices for the study area, computed from the GTOPO30 dataset. Blue and yellow shading correspond respectively to 'low' and 'high' values of each index.

- East-West slopes computed over 10km^2, 100km^2 and 1000km^2 squares. These were produced by regressing the mapped altitudes on the longitude co-ordinates within these squares, and converting the regression slope to units of (hundreds of metres' altitude change) per km. These slopes can be considered as a measure of "aspect", and are potentially useful for characterising (for example) rain shadow effects.
- North-South slopes, similarly.

Figure 3 provides maps of each of these indices over the study region.

1.1.3 Atmospheric predictors

We will consider two sets of atmospheric predictor data. Both are for the period September 1957 to August 2002, and are ultimately derived from the ERA40 Reanalysis dataset (http://apps.ecmwf.int/datasets/data/era40_daily/). Note that "official" VALUE work should use the ERA-INTERIM reanalysis rather than ERA40 (see http://www.value-cost.eu/validationTest). The reason for using ERA40 here is explained below. The atmospheric predictor sets are as follows:

- Spatially averaged values of selected variables, for the region 27.5°N–45°N, 10°W–15°E. The variables provided are mean sea level pressure, 10-metre u- and v- wind

velocity components and wind speed, 2-metre air temperature and dewpoint temperature. ERA40 provides 6-hourly fields of each variable (except wind speed, which is calculated from the u- and v-components) on a 2.5° grid; each variable has first been averaged over the selected region for each 6-hourly field, and then the results have been aggregated to provide daily and monthly time series for each variable.

This choice of variables has been guided by the results in **?**. These authors suggest that for downscaling in Spain one should use sea level pressure and mean temperature from their region Z8, which is the region considered here. In addition however, there is consensus within the experienced statistical downscaling community that some measures of (a) moisture availability (b) air flow may be helpful. This is why the dewpoint temperature and wind information are included.

- Daily and monthly time series of weather states for the Bay of Biscay and Iberian Peninsula, for the period September 1957 to August 2002. The data are derived from EU COST Action 733 (main web page at http://cost733.met.no/, although the data files and documentation are from http://cost733.geo.uni-augsburg.de/cost733wiki). COST733 compared many different weather classification techniques, for different European regions (see left panel of Figure 4). The region of interest here is region D09 "Iberia / W Med", formally defined as 31°–48°N and 17°W–9°E. From the available classifications produced by COST733, the data used here are based on the 8-state GWT (GrossWetterTypes) methodology derived from mean sea level pressure fields. This selection is based on advice from Radan Huth (personal communication), because the resulting states have a clear physical interpretation. Specifically, the classification is based on correlations between each day's MSLP field and selected "reference" fields, and reflects the rough direction of the flow as follows: 1 N, 2 NE, 3 E, 4 SE, 5 S, 6 SW, 7 W, 8 NW. For example, the right panel of Figure 4 shows the mean pressure field for all days classed as Type 1.

  One difficulty with the use of weather states for weather generation is that the statistical models can be parameter-intensive if there are many states. To simplify our task therefore, the initial set of eight GWT states has been reduced to five by hierarchical clustering: standardized seasonal anomalies in the temperature, proportion of wet days and wet-day precipitation intensity were calculated from the station data for each of the states, and states with similar values for each of these standardized anomalies were merged. Figure 5 shows clearly that states 1&2, 3&4 and 6&7 have similar precipitation-temperature climatologies. Thus the new state 1 can be interpreted as "N-NE flow", state 2 is "E-SE", state 3 is 'S', state 4 is "W-SW" and state 5 is "NW".

  The COST733 weather classifications are precomputed from ERA40 data, from 1957 to 2002. We will compare weather generators that use these weather types with those based on spatially-averaged predictors. To ensure a fair comparison, we must use the same time period and reanalysis product for all our weather generators. This is the reason for not using the "official" VALUE time period and reanalysis dataset.

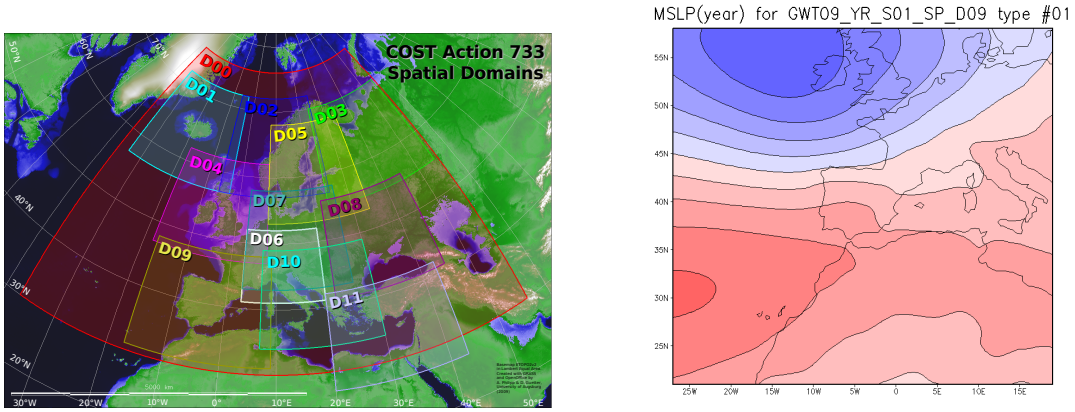

Figure 4: Left panel: European regions considered in COST Action 733 when assessing the performance of weather classification techniques. The region relevant to the present exercise is D09. Right panel: mean sea level pressure for the Bay of Biscay and Iberian Peninsula, for all days classed as type 1 (northerly) by the COST733 GWT classification scheme. The plot is from the COST733 web site at http://cost733.geo.uni-augsburg.de/cgi/cost733plot.cgi, which unfortunately does not provide any labels or explanation of the colour scales. The interpretation is therefore either of northerly flow (high pressure in the north and low pressure in the south) or southerly flow (vice versa).

1.2 Setting yourself up

You should have installed R and RStudio on your computers already, along with the necessary R packages — if not, follow the instructions in file `Rglimclim_Preparatory.pdf` at http://www.value-cost.eu/TS3. Having done this, everything you need for the practical sessions can be found in the `zip` archive `WeatherGenerators.zip` at the same web address. Download this archive, and unpack it somewhere sensible on your computer. It creates the following subdirectories:

`HANDOUTS` : contains copies of the handouts and slides for the weather generator training.

`PRACTICALS` : contains all of the material for the Rglimclim practical sessions.

`PREPARATORY` : contains the preparatory material that you were sent before the workshop, updated to include the latest version of Rglimclim.

There is an additional `zip` archive called `WGData.zip` on the workshop web page: this contains all of the original data as downloaded (except the ERA40 reanalysis data, for which a python script is provided so that you can retrieve the original data from the ERA40 servers). It also contains an R script called `makedata.r`, which was used to convert the original data into a format that Rglimclim can work with; and `README.txt` file to explain everything else. This archive is large (more than 30Mb) so you should not download it during the workshop. However, you may find it helpful later to discover how the processing was done.

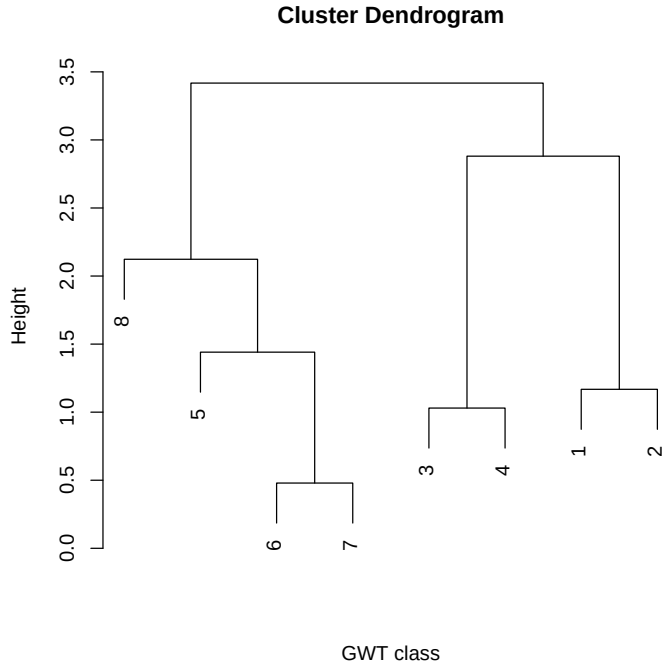

Figure 5: Hierarchical clustering of the GWT weather states from COST Action 733. Inter-state distances are computed from differences in mean standardized monthly anomalies for temperature, precipitation occurrence and wet-day precipitation intensity.

2 Session 1: building a weather generator

Our task in the first session is to build a bivariate weather generator that can generate simultaneous daily sequences of precipitation and temperature at any collection of locations in northern Iberia. This sounds difficult! Fortunately, Rglimclim is designed to help its users approach the task in a structured way; and, moreover, to avoid the tedious manipulation of data files that is often required when analysing multisite data.

Our weather generators are all based on generalised linear models (GLMs). We will use logistic regression models for precipitation occurrence, gamma distributions for precipitation intensity on "wet" days, and normal distributions for temperature. These choices are fairly standard for the modelling of precipitation and temperature in a GLM-based framework; see, for example, **???**. Specifically, let R_{st} and T_{st} denote the precipitation and temperature respectively, at site s on day t. Then the structure of our models is:

$$P\left(R_{st} > 0\right) = \pi_{st} \; ; \tag{1}$$

$$R_{st}|R_{st} > 0 \sim \Gamma\left(\alpha, \lambda_{st}\right) \quad \text{with} \quad \mathbb{E}\left(R_{st}|R_{st} > 0\right) = \alpha/\lambda_{st} = \mu_{st}^{(R)} \text{ , say;} \tag{2}$$

$$\text{and} \quad T_{st} \sim N\left(\mu_{st}^{(T)}, \sigma_{st}^2\right) \; . \tag{3}$$

The parameters π_{st} (the probability of experiencing precipitation at site s on day t, $\mu_{st}^{(R)}$ (the expected precipitation intensity on a 'wet' day), $\mu_{st}^{(T)}$ and σ_{st}^2 (the expected value

and variance of the temperature distribution) are related to linear predictors via logistic, logarithmic, identity and logarithmic link functions respectively. The parameter α (the coefficient of variation of precipitation intensity distributions) is assumed to be constant — this assumption has been found to be reasonable in the literature. Thus, denoting by $\eta_{st}^{(\cdot)}$ a generic linear predictor (i.e. linear combination of covariate values representing the effects of topography, seasonality, autocorrelation and large-scale atmospheric predictors), we set

$$\log\left(\frac{\pi_{st}}{1-\pi_{st}}\right) = \eta_{st}^{(\pi)} ; \tag{4}$$

$$\log \mu_{st}^{(R)} = \eta_{st}^{(\mu^{(R)})} ; \tag{5}$$

$$\mu_{st}^{(T)} = \eta_{st}^{(\mu^{(T)})} ; \tag{6}$$

$$\text{and} \quad \log \sigma_{st}^2 = \eta_{st}^{(\sigma^2)} . \tag{7}$$

We need to choose covariates and estimate their coefficients in each of these four different models. We also need to specify some inter-site dependence structures so as to build weather generators that are truly multi-site; however, this is fairly easy. We will proceed in three stages:

1. Build multi-site generators for each variable individually, without including atmospheric predictors. The aim is to characterise the climatology of each variable.
2. Link the individual generators to incorporate dependence between the two variables.
3. Extend the resulting joint generator by including the effects of large-scale predictors.

The first stage is by far the most difficult. I have therefore done it for you, and my 'baseline' models are provided to you as R objects. However, we will follow the process in detail for the precipitation occurrence model defined by equations (1) and (4).

We will build our weather generators using precipitation and temperature for the period 1960–1990; in tomorrow's session we will use the 1991–2002 period for validation purposes. Let's start.

2.1 Getting started

Start up RStudio, and use the `Session` menu to change the working directory to the `PRACTICALS/` directory created when you unpacked the `WeatherGenerators.zip` archive in Section 1.2. In the bottom right-hand subwindow, click on the file `ModelBuilding.r`: this is an R script containing all of the commands needed in the first part of this session. The script will open in the upper left-hand subwindow.

2.1.1 Loading Rglimclim

After the comments at the top of the script, the first command is `library(Rglimclim)`. This loads the Rglimclim package. Run the command: you should see the message

`Use 'help("Rglimclim-package")' to get started`. So: run the next command, which is `help("Rglimclim-package")`. Read through the text in the help window, noting in particular the idea of partitioning covariates into 'site', 'year', 'month' and 'day' effects and their interactions. When you have read through the help page, click one of the `PDF package manual` links. The package manual should open in an external window.[2] Scroll to page 15 of the manual, and then minimise its window — we will return to this later.

2.2 Reading topographic and station information

Run the next line in the R script: `load("Topography.rda")`. This loads the GTOPO30 topographic data for the study area, so that we can plot maps.

The next few commands read the site information from file `stations.dat`, discard any sites that are outside the study area and store the result in `station.data`:

```
LatLims <- c(42,44)
LongLims <- c(-9.5,0)
station.data <- read.table("stations.dat",header=TRUE,,
                           colClasses=c(rep("character",2),rep("numeric",16)))
station.data <- station.data[station.data$Latitude > LatLims[1] &
                             station.data$Latitude < LatLims[2] &
                             station.data$Longitude > LongLims[1] &
                             station.data$Longitude < LongLims[2],]
```

Next, we will identify some specific stations that have been chosen for inclusion in the VALUE validation test exercise (see file `stations_NorthIberia.pdf` in the archive `VALUE_validationTest_v2.zip` at http://www.value-cost.eu/validationTest):

```
station.data$VALUEstation <- rep(0,nrow(station.data))
station.data$VALUEstation[station.data$ID %in% c(" 420","1394","1395","3909",
                                                 "3910","3913","1392")] <- 1
```

We will use this information in Session 2, when we validate our weather generators.

I discovered one small problem with these EC&A data, when I first started using them. This is: there are two sites (3910 and 3911) that are recorded in identical locations, but whose records do not overlap. My guess is that in fact they are the same station with a change of recording equipment; however, the presence of two apparently different sites, at identical locations, creates numerical problems for Rglimclim (associated with singular covariance matrices). For our purposes, the easiest solution is to discard one of the sites, e.g. site 3911:

```
station.data <- station.data[station.data$ID != "3911",]
```

[2] This does not always work when using the package from within RStudio — there seems to be some conflict between RStudio and Acrobat Reader. If you get a "file is already open or in use" error, close down Acrobat Reader and open the file `manual.pdf` in the `PRACTICALS` directory instead. This is the latest version of the Rglimclim manual, extracted from the `doc/inst` folder of the `.zip` or `.tar.gz` archive from which you originally installed the package.

2.3 Defining the station information to Rglimclim

The next step is to convert all of this station information into a format that Rglimclim can interpret. This is done using the `make.siteinfo()` and `define.regions()` commands provided by Rglimclim. Run the next few lines of the script, up to

```
siteinfo <- make.siteinfo(station.data,coord.cols=c(4,3),
                          site.codes=1,site.names=2,region.col=19,
                          attr.names=site.attributes,
                          regions=define.regions(c("Northern Iberia",
                                                   "VALUE stations")))
```

This last command creates an Rglimclim site information database (formally, an *object of class* `siteinfo`) using the information held in `station.data`. Furthermore, specific columns of `station.data` have special interpretations, thus:

- Columns 4 and 3 respectively contain the x- and y- coordinates of the stations. This can be used to produce maps of spatial structure later on; also to calculate distances between stations, which will be useful.
- Column 1 contains *site codes*: these are 4-character codes used by Rglimclim to make the link between files containing predictand data, and the site information database. In this particular instance, the site codes are just the station numbers.
- Column 2 contains the site names: these will be used to label some of the output.
- Column 19 indicates which 'region' each station belongs to. A 'region', in Rglimclim, is a subset of sites for which we might want to compute simulation summary statistics. Regions are numbered consecutively starting with zero (the entire area — all stations are automatically assigned to this region). In our work, except for the entire study area, the only 'region' corresponds to the set of "official" VALUE stations. This 'region' is defined because we might want to study these stations in particular when assessing the performance of our weather generators in Session 2.
- The argument `attr.names` is used to set descriptive names for the site attributes. These will be used to produce informative labels for the output.

To see the effect of this, run the next command in the script: `print(siteinfo)`. You learn how many sites are in the database, as well as the attributes available for each site (to remind yourself what these attributes are, you may care to refer to Section 1.1.2). Notice that the order of the attributes has changed from the original data file: specifically, the longitude and latitude, as x- and y-coordinates, are now the first two attributes.

The `print()` command gives you an interpretable overview of the information contained within an object. In fact however, a `siteinfo` object contains a lot of detailed organisation, organised in a *list*. To see what lies beneath the surface, type the next command in the script which is `print(names(siteinfo))`. This is a useful trick: you

will often want to access information that is buried somewhere in an Rglimclim object, and the `names()` command can help you to find it.

This example of the `make.siteinfo()` command should be self-explanatory. If you want to find out more however, type `help(make.siteinfo)` at the console prompt.

2.4 Modelling precipitation occurrence

2.4.1 The simplest possible model

We have defined our site information; now we can fit some models. We will start with the simplest possible logistic regression model for precipitation occurrence, in which $\eta_{st}^{(\pi)}$ in equation 4 is a constant: $\eta_{st}^{(\pi)} = \beta_0$ say. In this case, our occurrence model is defined as

$$\ln\left(\frac{\pi_{st}}{1-\pi_{st}}\right) = \beta_0 \quad \text{or equivalently} \quad \pi_{st} = \frac{\exp[\beta_0]}{1+\exp[\beta_0]} \; . \tag{8}$$

To fit this model, we must first define its structure to the system and then estimate the parameter β_0. To help you, Rglimclim comes with an object already defined for this model structure, called `ConstantModel`. The next two lines in the script load the predefined Rglimclim objects (`data(GLCdemo)`) and display the structure of the "constant-only model" (`print(ConstantModel)`). They produce the following output:

```
CONSTANT-ONLY MODEL
===================
Main effects:
-------------
                                                    Coefficient
                                           Constant      0.0000

No dispersion parameters defined

Spatial dependence structure:
-----------------------------
Structure used: Independence
Warning message:
In print.GLC.modeldef(ConstantModel) :
No global quantities (trace thresholds etc.) defined
```

This is not particularly controversial: the coefficient value (β_0) is set to zero and there is some other information about dispersion parameters and spatial dependence, which is not important for the moment. We *should* worry, however, about the warning message. For many weather variables such as precipitation, the recording of small values may be inconsistent between stations and it is therefore common practice to define a threshold below which values should be considered as 'effectively zero'. No such threshold has been defined here, and Rglimclim warns us that this may cause problems. We *are* dealing with precipitation, so we should take the warning seriously. In the literature, a threshold of

1mm is often used to define a 'wet' day; so we will define a threshold of 0.95mm (this way, we guarantee that values of exactly 1mm are considered 'wet').

To define the 'trace threshold' to the system, we will define a revised model structure using a *model definition file*. The easiest way to do this is to create a valid definition file using the existing structure, and then to change it:

```
write.modeldef(ConstantModel,file="OccModel0Init.def")
```

This creates a file called `OccModel0Init.def`. You should see it in the `Files` tab of your lower right-hand subwindow in RStudio — if not, refresh the tab by clicking on the word `PRACTICALS` at the top of the tab. Now, proceed as follows:

1. Open `OccModel0_Init.def` by clicking in the `Files` tab.

2. Scroll down to the bottom of the file. Note the row containing the model title `CONSTANT-ONLY MODEL`: this will be used to label software output (it isn't very important, and only becomes useful when we're comparing different models). The only other part of the model definition is the final line (line 49 in the file)

   ```
   0    0.0000                Constant
   ```

   The first number here tells Rglimclim what *model component* is being defined — is it a site effect, a daily effect, an interaction, ...? Here the number is zero. Look at page 15 in the package manual that you opened earlier: component 0 corresponds to the constant term in the model. The second number is the coefficient value in the model definition — currently zero. The coefficient value doesn't matter at the moment — we are only defining the *structure* of the model.

3. Look again at Table 1 on page 15 of the Rglimclim manual. This shows that 'component 8' is used to define 'global' quantities, such as trace thresholds. It also refers us to Table 6, because the software needs to know how to handle 'trace' values if they occur. My experience is that 'soft thresholding' is often very effective (see reference list); this is option 2. So, add the line

   ```
   8    0.9500    1    2
   ```

   to the end of the `OccModel0_Init.def` file. This will be line 50. Take care with the alignment: the `8` should be below the `0`, the `0.1000` below the `0.0000` and the remaining values are each preceded by four spaces. Save the file and close it.

4. Read the new model definition into R (next line in the script):

   ```
   OccModel0.init <- read.modeldef("OccModel0Init.def",model.type="logistic",
                                   siteinfo=siteinfo,
                                   var.names=c("Precipitation","Temperature"))
   ```

You may be surprised that the model type, site information and variable names are specified at this stage. For more complicated models, these are sometimes needed to initialise the model structure. For more details on this command, type `help(read.modeldef)`.

If you get an error

```
Error in read.modeldef("OccModel0Init.def", model.type = "logistic", ...  :
  Input error while reading line 51 of file OccModel0Init.def.
```

the reason is that you have accidentally inserted a blank line at the end of the model definition file: re-open the file and delete it. The clue is in the error message — the software is trying to read line 51 of the file but, as noted earlier, the last line of model definition should be line 50.

5. Check that the system has interpreted the definition file as you intended, by running the next command `print(OccModel0.init)`:

```
CONSTANT-ONLY MODEL
===================

Response variable: Precipitation

Main effects:
-------------
                                                          Coefficient
                                                 Constant      0.0000


Global quantities:
------------------
     'Soft' threshold for +ve values:      0.9500

No dispersion parameters defined

Spatial dependence structure:
-----------------------------
Structure used: Independence
```

The software prints a meaningful description of the model, and the warning message has been replaced with the trace threshold definition.

6. Having defined the model structure to the system, we can fit the model:

```
OccModel0 <- GLCfit(OccModel0.init,which.response=1,siteinfo=siteinfo,
                    data.file="NorthIberiaPrecipTemp_1960-1990.dat",
                    diagnostics=1,nprev.required=-1)
```

The arguments to the `GLCfit()` command here are as follows:

`OccModel0.init` : this defines the model structure. Note that when we used `read.modeldef()` to create this object, we specified `model.type="logistic"`, so Rglimclim already knows that we want to fit a logistic regression model.

`which.response=1` : this tells Rglimclim that the *response variable* (predictand) is the first variable appearing in `data.file` (see below).

`siteinfo` : this provides information about the stations used in the analysis (in fact, for this very simple model, this information used only to identify the sites in the data file).

`data.file` : this is the name of the file containing the predictand data. For more details on the format of this file, type `help(GLCfit)` and scroll down to the `data.file` argument. In our example, `NorthIberiaPrecipTemp_1960-1990.dat` contains precipitation and temperature data from all of the EC&A stations in the study area for the period 1960 to 1990 (inclusive); precipitation is the first variable in the file, and temperature is the second.[3]

Don't worry about the other two arguments yet — we will return to them later.

The software responds with

```
Checking files ...
Reading data ...
Beginning estimation ...
      Iteration      Log-likelihood    Largest standardised score
      ---------      --------------    --------------------------
              0         -141815.141    -156.1451  (parameter   1)
              1         -129389.799      -6.2767  (parameter   1)
              2         -129370.036      -0.0311  (parameter   1)
              3         -129370.036      -0.0000  (parameter   1)
              4         -129370.036      -0.0000  (parameter   1)
Computing covariance matrix of estimates ...
```

The fitting is done by maximising a log-likelihood function obtained under the assumption that sites are independent of each other. The fitting algorithm is iterative: at each iteration, the routine outputs the current value of the log-likelihood as well as the largest standardised score (this is the gradient, normalised by the second derivative of the log-likelihood, for each parameter in the model).[4] In this case, convergence is fast: the maximised log-likelihood is -129370.036, and the gradient is zero so this is indeed a turning point of the log-likelihood function.

7. Inspect the fitted model by running the command `print(OccModel0)`:

```
CONSTANT-ONLY MODEL
===================
```

[3] The file was created using the `write.GLCdata()` command — see the `makedata.r` script in your `DATA/` subdirectory if you want more information.

[4] The amount of output can be controlled by the `verbosity` argument to `GLCfit()`.

```
Response variable: Precipitation

Main effects:
-------------
                                Coefficient  Std Err   Z-stat Pr(|Z|>z)
                          Constant     -0.7200   0.0137 -52.5457 < 2.2e-16

Global quantities:
------------------
     'Soft' threshold for +ve values:      0.9500

No dispersion parameters defined

Spatial dependence structure:
-----------------------------
Structure used: Independence
```

Notice the following: :

- The 'Coefficient' value of -0.7200 is the estimate of β_0 in (8). The corresponding probability of precipitation on any day is $\exp(0. - 0.72)/[1 + \exp(-0.72)] = 0.327$. This is a complicated way to learn that 32.7% of the precipitation values in the database are greater than 0.95mm, but it provides some insight into the model structure, and serves as a simple check on the output!
- The value of 0.0137 in the 'Std Err' column is a robust standard error that accounts for the possibility that data from different sites are not independent.[5] The associated Z-statistic and p-value allow us to test the null hypothesis $H_0 : \beta_0 = 0$ against the alternative $H_1 : \beta_1 \neq 0$: the tiny p-value suggests an overwhelming rejection of this hypothesis which is, however, of no scientific relevance for this particular model.

2.4.2 Checking the model

In the `GLCfit()` command above, we used the argument `diagnostics=1`. This tells the routine to store some basic information to help check the model structure (alternatives are `diagnostics=0` to suppress these calculations, and `diagnostics=2` to produce an

[5] All "standard" GLM fitting packages calculate their estimates and standard errors under the assumption that the observations are independent. This assumption is almost certainly false when fitting to multi-site time series data — in particular, data from neighbouring sites are unlikely to be independent. However, it can be shown that the estimates themselves will still be reasonable even in the presence of such dependence — so Rglimclim is justified in using an "independence log-likelihood" to fit the model. A more serious consequence of inter-site dependence is that the usual standard errors for the parameter estimates are incorrect. To see the standard error that would be obtained from a "standard" GLM software routine, type `print(OccModel0,which.se="naive")`. You will find that the correct standard error is almost three times as large as this "naïve" version — so uncertainty in the parameter estimates could be seriously underestimated unless inter-site dependence is accounted for.

output file containing detailed diagnostic information for further analysis). To see some of these diagnostics, type `summary(OccModel0,tables=NULL)`. Don't worry about the `tables` argument for the moment.

You will see some basic information about the fit, followed by a table entitled `Occurrence frequencies vs forecasts`. The idea behind this table is that if we collect together all days for which the modelled probability of precipitation is 0.1, then 10% of these days should have experienced precipitation. In practice, we collect together all days for which modelled probabilities lie in the ranges $(0, 0.1), [0.1, 0.2), \ldots, [0.9, 1.0)$ and calculate the observed and expected numbers of wet days in each case. A lack of agreement in any cell of the table indicates a problem with the model. Clearly however, for the present trivial model the information from such a table is not particularly useful (it indicates that there were 204596 days — i.e. all of them, because the model only contains a constant term — when the modelled probability of precipitation was between 0.3 and 0.4, and that the observed and expected proportions of wet days were both 0.327).

As well as checking the probability structure, it is useful to check for systematic structure in the data that is not captured by the model. Graphical diagnostics provide an easy way to to this. Run the next few commands in the script:

```
if (dev.cur()==1) x11(width=8,height=6)
par(mfrow=c(2,2),mar=c(3,3,3,2),mgp=c(2,0.75,0),lwd=2,ask=TRUE)
plot(OccModel0,which.plots=1:2,lwd=2)
```

The first of these commands opens a new graphics window, unless a graphics device is open already. The next command sets up a 2×2 array of plots, with some control over the margins and line widths, and the final `plot()` command produces the diagnostic plots.

The first plot is a graphical representation of monthly mean *Pearson residuals* from the model. If the model is correct, the individual Pearson residuals all come from distributions with mean zero and the same standard deviation (usually 1). The monthly mean Pearson residuals should therefore be randomly scattered around zero: the dashed lines in the plot show the range within which most of them should lie. Here, most of the mean residuals fall outside the dashed lines, with a very strong seasonal cycle: precipitation occurrence is less frequent in summer, and more frequent in winter, than the model predicts.

The second plot shows the monthly standard deviations of the Pearson residuals, along with their expected value of 1. The plots in the bottom row show the same information but for annual rather than monthly means: these are designed to reveal any time trends that have not been captured by the model. Here, there is some seasonal structure in the monthly standard deviations, and also a sharp decline in precipitation occurrence at the end of the record (large negative mean residuals in 1989 and 1990).

The clear structure in these plots tells us that the model is inadequate. An obvious way to improve things is to model the seasonality. Before we do this however, it may also be helpful to look for unexplained *spatial* structure:

```
par(mfrow=c(1,1),mar=c(5,3,5,3))
image(topo.longs,topo.lats,topo.data,col=terrain.colors(50),axes=FALSE,
      xlab="",ylab="")
```

```
box(lwd=2)
plot(OccModel0,which.plots=3,
     site.options=list(add.to.map=TRUE,site.labels="none",scale=4))
```

This produces a topographic map of the study area, with circles drawn at each station location. The relative sizes of the circles indicate the values of the standardised mean Pearson residuals at each site; the solid (dashed) circles indicate sites with positive (negative) mean residuals i.e. sites that are wetter (drier) than the model. Circles drawn in 'thick' lines indicate mean residuals that differ significantly from zero at the 5% level. This occurs here at almost all sites. Moreover, the solid circles tend to be around the coast and the dashed circles are inland: this clear spatial organisation suggests that there is systematic regional variation in the precipitation occurrence (coastal sites experience more frequent precipitation than inland sites), which is not captured by the model.

Notice the differences between the two `plot` commands above: the first one used the arguments `which.plots=1:2` for seasonal and annual diagnostics, and the second used `which.plots=3` for regional diagnostics. For more details, type `help(GLC.modeldef)`.

2.4.3 Including seasonality

We have established that there is unmodelled seasonality in the data(!). We will therefore add some seasonal structure to the model. The steps are as follows:

1. Write the fitted model structure to a new definition file called `OccModel1Init.def`:

   ```
   write.modeldef(OccModel0,file="OccModel1Init.def")
   ```

2. Open this new definition file for editing. Notice that the value of the 'Constant' term is now -0.7200, as fitted previously.

3. Choose a plausible representation of seasonality, from the options available in Tables 1 and 2 on pages 15 and 16 of the Rglimclim manual. A good starting point is usually a Fourier representation, at a daily timescale. This requires both cosine and sine coefficients, since the phase of the cycle is unknown. For a daily timescale, we must look at Table 2. This tells us that the required terms correspond to 'Code 1' values of 21 and 22 respectively. So to define a simple seasonal cycle, insert the following two lines between the 'Constant' and 'Trace threshold' rows (note that in Rglimclim model definition files, rows must be ordered according to the 'component' value which is 4 for daily effects and 8 for the trace threshold):

   ```
   4    0.0000   21
   4    0.0000   22
   ```

   In the absence of prior information, zero is often a good initial value for the coefficients in model definition files.

   Once again, take care with the alignment: there should be three spaces before both the `21` and `22` so that these codes each occupy five positions in their respective records, and so that they align with the first code in the 'trace threshold' definition.

   Save the modified definition file and close it.

4. Read the new model definition into R and check it:

```
OccModel1.init <- read.modeldef("OccModel1Init.def",model.type="logistic",
                                siteinfo=siteinfo,
                                var.names=c("Precipitation","Temperature"))
print(OccModel1.init)
```

Under `Main effects`, you should now see the following (if not, you have made a mistake and should correct the definition file before proceeding):

```
                                                    Coefficient
                                          Constant      -0.7200
1                 Daily seasonal effect, cosine component       0.0000
2                   Daily seasonal effect, sine component       0.0000
```

5. Fit the updated model and inspect it:

```
OccModel1 <- GLCfit(OccModel1.init,which.response=1,siteinfo=siteinfo,
                    data.file="NorthIberiaPrecipTemp_1960-1990.dat",
                    diagnostics=1,nprev.required=-1)
print(OccModel1)
summary(OccModel1,tables=NULL)
```

In this new model, all three coefficients differ from zero at any reasonable level of significance (this is not surprising, because we known that precipitation is seasonal). The `Occurrence frequencies vs forecasts` table now shows some variation in the modelled precipitation probabilities, coming from the modelled seasonal cycle. To see whether the model captures all of the seasonality in the data, plot the results again:

```
par(mfrow=c(2,2),mar=c(3,3,3,2),mgp=c(2,0.75,0),lwd=2,ask=TRUE)
plot(OccModel1,which.plots=1:2,lwd=2)
```

This still shows some seasonal structure in the monthly mean Pearson residuals, with peaks in April–May and October–November and a minimum in July. This kind of structure, where the peaks are separated by a period of about six months, suggests that the seasonal cycle is not an exact sinusoid. A simple remedy is to add a harmonic term to the model — Rglimclim (Table 2 on page 16 of the manual) allows this via codes 23 and 24. Accordingly, create a new model definition file:

```
write.modeldef(OccModel1,file="OccModel2Init.def")
```

Open the file for editing, and insert two new rows so that the main model definition section looks like this:

```
  0   -0.7438              Constant
  4    0.4488   21         Daily seasonal effect, cosine component   1
  4    0.2674   22         Daily seasonal effect, sine component     2
  4    0.0000   23
  4    0.0000   24
  8    0.9500    1    2    'Soft' threshold for +ve values
```

Save the file, close it, read the updated model definition, check it, fit the new model, print the result, view the diagnostic table and produce the diagnostic plots (script commands up to `plot(OccModel2,which.plots=1:2,lwd=2)`). You should see that most of the systematic seasonal structure (in both the mean and standard deviation) has been eliminated. There is a large negative mean residual in March, but we should bear in mind that the dashed lines in the plots are computed under the assumption that the model is correct, and this is certainly not the case (we haven't modelled the regional variation yet, nor have we attempted to account for autocorrelation). We should revisit this later. Similarly, the strong downward trend at the end of the 1980s can perhaps be explained later by including the effects of large-scale atmospheric predictors in the model.

For our initial, constant-only model, the final residual plot provided evidence of systematic regional variation in precipitation occurrence. Run the next set of commands, from `par(mfrow=c(1,1),mar=c(5,3,5,3))` to `plot(OccModel2,which.plots=3, ...`, to regenerate the same plot for the new model. Unsurprisingly, nothing has changed. We will therefore try to model the regional variation next.

2.4.4 Systematic regional variation

The representation of systematic regional variation is often the most difficult task when developing a multi-site weather generator. Rglimclim offers some fairly simple, but nonetheless flexible options. In this particular instance, the pattern of positive and negative mean residuals could perhaps be approximated using a quadratic surface, with a minimum in the south-east of the study area and increasing up towards the north and west. Such a quadratic surface may be represented by adding linear and quadratic functions of latitude and longitude to the model, along with a cross-product of the linear terms (i.e. x_1, x_1^2, x_2, x_2^2 and $x_1 x_2$, where x_1 and x_2 denote latitude and longitude). In the GLM framework, the cross-product term can formally be regarded as an *interaction* between the latitude and longitude effects (more on interactions later).

Instead of working directly with latitudes and longitudes here however, we will use *Legendre polynomials*. You don't need to know much about these — just think of them as a useful reparameterisation of the usual polynomials. The motivation for using them is that they are approximately uncorrelated if the stations are spread roughly uniformly across the study area (**?**): this increases the computational stability of the model fitting and avoids excessive collinearity between different covariates in the model.

Legendre polynomials are always defined over a finite range. We must therefore specify the range of latitudes and longitudes that we are interested in. We will just use the definition of the study area i.e. 42°N–44°N, 9.5°W–0°W.

If you were doing this yourselves, at this point you would need to write the existing model definition to a new file (`write.modeldef(OccModel2,file="OccModel3Init.def")`) and then edit it. From here onwards however, to save time I recommend that you just use the definition files that have already been prepared for you. These can be found in the subdirectory `PRACTICALS/DEFINITIONS/`: you may care to copy all files named `OccModel?Init.def` to `PRACTICALS/`so that you can see them. The amended version of `OccModel3Init.def` contains the following:

```
0   -0.7530                 Constant
1    0.0000    1   31
1    0.0000    1   32
1    0.0000    2   31
1    0.0000    2   32
4    0.4696   21         Daily seasonal effect, cosine component   1
4    0.2795   22         Daily seasonal effect, sine component     2
4   -0.1113   23         Daily half-year cycle, cosine component   3
4   -0.1867   24         Daily half-year cycle, sine component     4
5    0.0000    1    3
7   -9.5000    1    1
7    0.0000    1    2
7   42.0000    3    1
7   44.0000    3    2
8    0.9500    1    2     'Soft' threshold for +ve values
```

Some explanation would be helpful:

- The first four new rows have the "component" field set to 1 — from Table 1 on page 15 of the manual, this indicates that they correspond to site effects.
- There are two 'codes' on each of these four rows. The first selects the site attribute, as defined in the `siteinfo` object (recall that the first two attributes are longitude and latitude, respectively). The second defines a transformation of the site attribute. From Table 3 on page 17 of the manual, linear and quadratic Legendre polynomials are transformations 31 and 32. So: these four rows together define linear and quadratic Legendre polynomial transformations of latitude and longitude.
- The next addition to the definition file is the row with the "component" field set to 5. From Table 1 in the manual, this defines a two-way interaction term; and the 'Code 1' and 'Code 2' select the interacting covariates. In our updated model definition, the linear terms in longitude and latitude are respectively the first and third covariates (not including the constant term).
- The final additions have the "component" field set to 7. Again from Table 1, these define parameters in nonlinear transformations. Table 3 shows that a Legendre polynomial transformation requires two parameters, defining the range for the representation. Accordingly, the first (minimum) and second (maximum) parameters in the Legendre polynomial transformation of covariate 1 (longitude) are defined as -9.5 and 0.0 respectively; and the corresponding latitude (covariate 3) limits are set to 42 and 44. Rglimclim is smart enough to realise that the same limits should be applied to covariates 2 and 4 (the quadratic terms).

Read this new model definition using `read.modeldef()` and print the result to check that it is correct. Fit the model and look at the residual map (script commands up to `plot(OccModel3,which.plots=3, ...)`. There is much less systematic structure now, although there are still many thick circles. There are two sites in the south-west of the study area with *very* large circles. It might be worth finding out more about these:

```
summary(OccModel3,tables="site")
```

By specifying `tables="site"`, we obtain a table with details of the residual performance at each site individually. Scanning through this table, the mean residuals at Ourense (Instituto) and Pontevedra (Instituto) seem particularly large. Perhaps there is a problem with the data from these stations? Alternatively there may be some topographic controls on precipitation occurrence at these two sites.

To explore this possibility, we can plot the mean Pearson residuals at all sites against each of the topographic indices in our `siteinfo` object. To do this, we must find out how to access the residual information. The next few lines of the script (up to `site.resids <- OccModel3$Residuals$Pearson$Site.table$Mean`) show how to do this, and to extract the mean site residuals to a vector that can be used for plotting. The next command uses `all.equal()` to check that the residual table is in the same order as the site information; and then, after setting up a 4×4 array on the graphics device, the loop `for (i in 3:16) { ... }` produces the required plots.

The two sites with large mean residuals appear as outliers on every plot. Thus the large mean residuals cannot obviously be linked to any of our topographic indices. Moreover, their inclusion in the model could seriously distort the estimated topographic effects for the remaining sites (e.g. they contribute substantially to the apparent negative correlation between altitude and precipitation occurrence).

We have two options: either discard the data from these two sites, or account for them in the model. In general, it is poor scientific practice to discard data without good reason: the problem could be with the model, not the data. Therefore, in the first instance we will account explicitly for these sites in the model. The easiest way to do this is to add a 'dummy' site attribute, taking the value 1 at each of these two sites and 0 at the remainder. If this attribute is added to the model as a covariate, its coefficient will provide an adjustment to the linear predictors $\{\eta_{st}^{(\pi)}\}$ at these two sites alone.

In the script, the next few lines of code redefine the `siteinfo` object to include this additional 'dummy' attribute. An updated model definition file, `OccModel4Init.def`, is then produced incorporating (again: from here onwards you should use the version that has already been prepared for you). The updated model is fitted, and the results and diagnostics are viewed (command `summary(OccModel4,tables=NULL)`).

Look carefully at the `Occurrence frequencies vs forecasts` now. Notice that with this new model, there are $6 + 563 + 2452 = 3021$ cases for which the modelled probability of precipitation exceeds 0.8; and the observed proportion of 'wet' days is between 0.87 and 1 for these groups of cases. If you check the summary table from `OccModel3`, you will find that the two outlying sites between them contributed 3021 observations to the database. Thus, the records from these sites suggest that precipitation occurs on around 90% of days. This is clearly wrong (perhaps the observers at these stations tend to report only non-zero precipitation values?). The data from these two sites must therefore be considered unreliable, and we are justified in discarding them.

The easiest way to discard the sites is to remove them from the `siteinfo` object using the next two commands in the script. Now, Rglimclim will ignore their data because it no longer recognises them in the data file (we do *not* need to change the data file itself). Next,

Model 3 is refitted (this was the model that did not include the 'outlying site indicator') and the result is stored as `OccModel5`. The `summary` and `plot` commands now suggest that there are no longer any very large mean Pearson residuals — although most of them are still significantly different from zero according to the model.

Now we can look at the effect of topography again. The next few commands recompute the mean Pearson residuals at each site, for the new model; and plot them against each of the topographic indices. We might ask: what is the relevant spatial scale for each of the main categories of topographic predictor (altitude, topographic variability and slopes)? From the correlations on the plots, the mean Pearson residuals are (slightly) more strongly correlated with the station altitude than with any of the altitude indices derived from the GTOPO30 data (notice also that the correlation is now positive, whereas it was negative when the outlying stations were included). Also, for topographic variability the correlations are highest at the 1000km^2 scale, and for the slope variables the correlations are highest at the 10km^2 scale. So we add these four site attributes, to obtain `OccModel6` (note that in the updated version of `siteinfo`, the required attributes are numbers 3, 10, 11 and 14 — type `siteinfo` to verify this). When we `print(OccModel6)`, the north-south slope seems insignificant. However, we will keep it in the model for the moment, because it may become important in conjunction with circulation indices later on.

The `summary` and `plot` commands for `OccModel6` suggest that the mean Pearson residuals at each site are generally small in absolute value, although many of them remain significantly different from zero (note that the circles in the plot seem *larger* than before, but this is an artefact of the way that Rglimclim tries to find a sensible way of scaling the circles, not always successfully — do not worry about this, therefore![6]).

We have captured a reasonable amount of the systematic regional structure now; and our modelling has also revealed some data errors. Next, we look at autocorrelation.

2.4.5 Accounting for autocorrelation

The modelling of autocorrelation (i.e. temporal, rather than spatial dependence) is achieved in Rglimclim by including previous days' values as covariates in a model. This poses several questions, for example: how many previous days' values should be included? Should we transform them? If so, how? Should we consider previous days' values at each site individually, or is it better to use averages over some neighbourhood of each site?

Many weather generators use a first-order Markov chain to represent autocorrelation in precipitation occurrence. Moreover, in simple weather generators it is common to fit separate Markov chains in different seasons, or in different months of the year, so that the autocorrelation varies seasonally. In a GLM, a first-order Markov structure can be incorporated by including, for each observation, a covariate taking the value 1 if precipitation occurred on the previous day at the same site, and zero otherwise. Seasonal variation can be incorporated by defining an interaction between the 'previous day' and 'seasonal' covariates.

The initial definition file for our next model, `OccModel7aInit.def`, illustrates how

[6] If you want to feel happier, set `scale=1` in the plot command ...

this is defined in Rglimclim:

```
...
    4    0.4640   21          Daily seasonal effect, cosine component   9
    4    0.2805   22          Daily seasonal effect, sine component    10
    4   -0.1089   23          Daily half-year cycle, cosine component  11
    4   -0.1871   24          Daily half-year cycle, sine component    12
    4    0.0000    1    3
    5    0.7195    1    3     2-way interaction: covariates  1 and  3
    5    0.0000    9   13
    5    0.0000   10   13
    7   -9.5000    1    1    0Parameter 1 in transformation of covariate 1
...
```

Note the following:

1. There is a new 'component 4' row (recall that component 4 indicates a daily effect). The first code is 1 and the second code is 3. Table 2 in the manual shows that the '1' indicates a value taken from 1 day previously; and that in this case, the second code selects a transformation. Table 4 (page 18 of the manual) now tells us that transformation 3 corresponds to an indicator taking the value 1 if the previous day was wet and 0 otherwise.

2. There are two new 'component 5' rows for defining two-way interactions. In the first instance we will include interactions between the previous day's occurrence indicator and the main sine and cosine terms for seasonality (excluding the harmonics). The interpretation of this is that the coefficient of the previous day's occurrence itself follows a sinusoid through the year. To define these interactions, we must define the indices of the interacting *main effects*. At this point, notice the numbers that Rglimclim has inserted at the end of each line when writing model definition files: we have not inserted any new lines *before* the covariates that we're currently interested in, so these are the required indices. Thus, the cosine and sine components of seasonality are covariates 9 and 10; and the new 'lagged' covariate is number 13. So we add interactions between covariates 9 and 13, and 10 and 13, to the model.

Read the definition file and inspect the result (`print(OccModel7a.init)`). Notice the line

```
13                                  I(Precipitation[t-1]>0)      0.0000
```

Rglimclim constructs informative covariate labels from the model definitions: read them carefully to check that your definitions are correct!

The next command fits the model, as usual, to produce `OccModel7a`. However, this time there is a difference: notice the argument `nprev.required=4`. This is because we will soon compare models containing 2, 3 and 4 previous days' values, to find out how many previous days' values are needed. However, to compare models formally they must

be fitted to the same data. A model containing 4 previous days' values can only be fitted to observations with 4 previous days' values available: by specifying `nprev.required=4`, we ensure that *all* of our models are fitted to this subset of the observations.

Before looking at the fitted model, we can explore some other options for modelling autocorrelation. For example: instead of considering the previous day's value individually at each site, perhaps it would be better to consider the average of these previous day's indicator variables (i.e. the proportion of sites experiencing rain)? To implement this (definition file `OccModel7bInit.def`), just change the second code in the 'previous day' line of the model definition file from 3 to 13 (see Table 4 in the manual again). This time, when you read the model definition you will see a warning message — this is directed at users of the original GlimClim package (the predecessor to Rglimclim) so we should not worry about it. Check the definition (`print(OccModel7b.init)`) and fit the model.

A third possibility is to consider a *weighted* average of the previous days' indicator variables, where the weights decay exponentially with distance from the site of interest. This formulation in fact includes both of the previous models as special cases: if the decay rate is zero then the weights are all equal and we recover the unweighted average used in `OccModel7b`, whereas as the decay rate increases indefinitely then, ultimately, all of the weight is concentrated on the site of interest and we recover `OccModel7a`. Rglimclim will find the maximum likelihood estimate of the decay rate, along with all of the other model parameters. This is a challenging numerical problem, however, and it is helpful to provide good starting values. Fortunately, these can be taken from `OccModel7b`, because this corresponds to a model with a decay rate of zero. Accordingly, the definition file `OccModel7cInit.def` is created from `OccModel7b`, with 23 instead of 13 to choose the "exponentially weighted average" transformation, and with an additional line

```
7    0.0000   13    1    1
```

in the 'nonlinear parameters' section (component 7). The parameter here is the exponential decay rate (see Table 5 on page 21 of the manual), and is set to zero because this corresponds exactly to the fit of `OccModel7b`. The '13' refers to the covariate number (notice that the 'previous day' covariate is number 13) and the next '1' indicates the first parameter in the nonlinear transformation — in exactly the same way that we previously defined the limits of the Legendre polynomial representations of latitude and longitude. The final '1' is new, however: this tells Rglimclim to estimate this nonlinear parameter from the data (the corresponding code is zero for all of the Legendre polynomial parameters, so Rglimclim considers these to be fixed and known).

In the specimen script, the `read.modeldef()` command for `OccModel7c` includes a new argument: `oldGlimClim.warning=FALSE`. This suppresses the annoying warning message.

When you inspect the new model definition (`print(OccModel7c.init)`), you will see a message telling you that the distance-dependent weights are based on distances calculated from longitude and latitude (these are the first two attributes defined in `siteinfo`, corresponding to the `coord.cols` argument when we used `make.siteinfo` earlier). Rglimclim always computes inter-site distances in this way.

`GLCfit()` for this model is noticeably slower than before, and it requires many more iterations to converge. This is caused by estimation of the decay rate in the weighting

scheme, which enters nonlinearly into the linear predictors $\{\eta_{st}^{(\pi)}\}$ and complicates the numerical optimisation problem considerably.

We have now fitted three different models incorporating dependence on the previous day's precipitation occurrence, and also allowing the strength of this dependence to vary seasonally. Which of these models provides the best fit to the data? One way to answer this is to look at the independence log-likelihoods for the fitted models: the `logLik()` command can be used to do this, as illustrated in the next three script commands. The log-likelihood increases considerably from `OccModel7a` to `OccModel7b`, and again from `OccModel7b` to `OccModel7c`. `OccModel7a` and `OccModel7b` have the same number of parameters, so we can safely conclude that it is better to model autocorrelation using the proportion of wet sites than using the simple Markov structure. The third model has one extra parameter, so some increase in log-likelihood is to be expected. However, an increase of around 1626 is much bigger than would normally be expected for the addition of a single parameter; and, moreover, if you type `print(OccModel7c)` you will see that the estimated decay rate is 0.5538 with a (robust) standard error of 0.0209. An approximate 95% confidence interval for the decay rate is therefore $0.55 \pm (2 \times 0.02) = (0.51, 0.59)$ which excludes zero. There seems to be convincing evidence, therefore, that the decay rate is not zero; and hence we should prefer `OccModel7c` over `OccModel7b`.

Having established that autocorrelation is most plausibly modelled using a weighted average of previous days' values, we now turn to the question: *how many* previous days' values? We can answer this by fitting models involving two (`OccModel7d`), three (`OccModel7e`) and four (`OccModel7f`) previous days' values. We retain the weighted average for all of these; note that Rglimclim uses the same decay rate for each previous day, so that the 'decay rate' parameter only needs to be defined once. Experience also suggests that there is limited evidence for interactions between autocorrelation and seasonality at lags higher than one day; so these interactions are not considered here. However, it *is* now important to give descriptive titles to the models (you may have noticed that all of our definition files so far contained the text `CONSTANT-ONLY MODEL`, which was a bit lazy). To see why, fit all of the models and then run the next command in the script: `anova(OccModel7c,OccModel7d,OccModel7e,OccModel7f)`. The result is as follows:

```
Comparison of nested models
---------------------------

Model 1: MODEL WITH WEIGHTED AVERAGE OF 4 PREVIOUS DAYS' PRECIP OCCURRENCE
Model 2: MODEL WITH WEIGHTED AVERAGE OF 3 PREVIOUS DAYS' PRECIP OCCURRENCE
Model 3: MODEL WITH WEIGHTED AVERAGE OF 2 PREVIOUS DAYS' PRECIP OCCURRENCE
Model 4: MODEL WITH WEIGHTED AVERAGE OF 1 PREVIOUS DAY'S PRECIP OCCURRENCE

         Resid DF DF2-DF1      LogL     LLR          p Robust LLR   Robust p
M1         196807         -103185.1
M1 vs M2   196808       1 -103197.9  12.725 4.5389e-07      2.129   0.039057
M2 vs M3   196809       1 -103365.0 167.109 < 2.22e-16     26.715 2.6787e-13
M3 vs M4   196810       1 -103454.3  89.342 < 2.22e-16      12.37 6.5636e-07
```

This table provides a formal *comparison of nested models*. The models are ordered

in decreasing order of complexity and, at each stage, a hypothesis test is performed: the null hypothesis is that the data were generated from the simpler of the two models. The test is based on the difference in the maximised log-likelihoods at each stage (column `LLR` in the output). In the absence of residual inter-site dependence, standard theory can be used to derive a p-value for the test (column `p`), taking into account the number of additional parameters in the more complex model (column `DF2-DF1`). However, in the present setting it is likely that there is unmodelled inter-site dependence: therefore we should adjust the difference in log-likelihoods (`Robust LLR`) to obtain an adjusted p-value (`Robust p`). The adjustments use the theory developed in `?`.

The conclusions from this analysis are:

- The test for `M3` versus `M4` leads to a convincing rejection of `M4`, with a robust p-value of 6.56×10^{-7}: this suggests that more than one previous day's value is needed in the model (now you see the reason to provide descriptive titles for the models!).
- The test for `M2` versus `M3` leads to an overwhelming rejection of `M3`, similarly: more than two previous days' values are needed.
- With a robust p-value of around 0.04, the test for `M3` versus `M4` leads to a rejection of M3 at the 5% level. Bear in mind, however, that we are fitting models to around 200,000 observations. With a dataset of this size, very small effects, that are of no practical relevance, can appear statistically significant. Experience suggests that for a dataset of this size we probably should not get too excited by p-values unless they are smaller than about 10^{-3}, unless there are good reasons to retain the corresponding terms in the model (e.g. the potential for a north-south slope variable to interact with circulation indices, as noted above).

So: at this stage, `OccModel7e` seems to be the model of choice. Examine the estimated coefficients, the summary, and the plots. The script provides some comments on the results — do you agree with them?

2.4.6 Finalising the baseline occurrence model

We have now found reasonable representations of seasonality, regional variation and autocorrelation. Notice, however, that `OccModel7e` was fitted only to observations for which four previous days' values were available: now that we have decided to use only three previous days' values, we could perhaps increase the precision of our estimates by refitting the model to take advantage of some observations that were excluded from the previous fit. At the same time, we can add a few additional interaction terms, representing the possibility that both the seasonal cycle and the 'lag 1 autocorrelation' coefficient may vary over the study area (we will consider only the linear polynomial terms in longitude and latitude here, to keep things relatively simple). Thus we want interactions between the covariate pairs $(1, 9)$, $(1, 10)$, $(2, 9)$, $(2, 10)$, $(1, 13)$ and $(3, 13)$.

There is a further critical feature that is missing from our model. So far, except for the calculation of robust standard errors and adjusted likelihood ratios, our modelling

has not addressed the issue of potential dependence between sites (you may have noticed, in the software output, that the sites are assumed to be independent). This will cause problems if we try and simulate precipitation sequences from the fitted model in its current form, since the simulated sequences from each of the sites will be independent except for dependence induced via the covariates. This is probably unrealistic.

Rglimclim offers several alternative ways to model inter-site dependence: some of these are specific to precipitation occurrence modelling. In the present setting, the study area is fairly large and it is therefore likely that dependence between nearby sites will be much stronger than dependence between distant sites.[7] In such situations, it is convenient to model the dependence via correlated latent Gaussian variables as described on page 9 of the Rglimclim manual; and, in the first instance, it is often sensible to start by considering the inter-site correlations to decay exponentially with inter-site distance. Referring to Tables 1 and 7 of the manual, this correlation structure is defined by adding the line

```
10    0.0000    3    1
```

to the end of the definition file. The component value of 10 refers to the residual inter-site dependence; the parameter value of zero is an initial value (which is ignored by the software, in fact); the '3' selects the exponential correlation function in Table 7; and the '1' indicates that we are defining the first (and only) parameter in this function.

The required definition file is `OccModel8Init.def`. Read this, check the definition and fit the model. Notice the argument `cor.file="OccModel_Corr.dat"` to the `GLCfit()` command this time: the inter-site correlation estimates for the latent Gaussian variables will be written to this file (Rglimclim tries not to store large objects internally, because this can cause memory problems within R itself). The correlations take some 30–40 seconds to estimate. You may notice a warning, telling you that the observed joint occurrence frequency is incompatible with the model at one pair of sites: this is probably unimportant (if you want to know more about this, see Appendix E.3.1 of the Rglimclim manual).

Finally, inspect the fitted model, the summary, and the residual plots. There is one extra plot this time (`plot(OccModel8,which.plots=5,plot.cols=c("blue","purple"))`), to check the fit of the inter-site dependence model. The 'data points' in this plot represent the estimated correlations between latent Gaussian variables at each pair of sites; the intensity of shading indicates the sample size available for each correlation estimate, with larger samples corresponding to more intense shading. The fitted curve seems to provide a reasonable fit overall. The only matter for concern is a group of negative correlations which, on inspection, are all associated with a single station. If we had more time, perhaps we would investigate this further. For the moment however, we will take this as our final 'climatology' model for precipitation occurrence.

2.5 Models for precipitation intensity and temperature

You have now seen how to approach the model-building process in a structured and informed way, using the Rglimclim diagnostics to suggest model improvements and identify

[7] This contrasts with the situation in very small catchments, where inter-site dependence is often uniformly high.

potential data errors. Although this process seems intimidating at first, with experience it becomes relatively straightforward: an experienced user can build such a model in one or two hours. You will be pleased to learn, however, that we will not repeat the same model-building process to develop 'climatology' models for precipitation intensity and temperature: these have already been prepared for you and are stored in files `Intensity.rda` and `Temperature.rda`. Moreover, the final occurrence model is stored in file `Occurrence.rda`. If you are interested, the sequence of analyses used to build these models is provided in the remainder of the `ModelBuilding.r` script.

We do not need `ModelBuilding.r` any more, unless you want to refer to it to check the syntax for some of the commands. If you want a record of your commands from here on, you should probably start a new script ("New File" on the "File" menu in RStudio).

At this stage you may also find it helpful to empty your R workspace and start afresh. You don't have to do this — it is just to remove a large number of objects that you no longer need, but you may want to keep them anyway. The command to empty your R workspace is `rm(list=ls())`. You can then use

```
load("Occurrence.rda")
load("Intensity.rda")
load("Temperature.rda")
load("Topography.rda")
```

to restore all of the objects that you *will* need. These are `OccModel8` (the precipitation occurrence model that we have just developed), `IntModel5` (a precipitation intensity model based on gamma distributions), `TempModel8` (a temperature model based on normal distributions), and our old friends `siteinfo`, `topo.data`, `topo.lats` and `topo.longs`.

Look at models `IntModel5` and `TempModel8`, to see their structure. It is similar to the structure of the precipitation occurrence model, although there are differences in the details (e.g. autocorrelation is modelled using different transformations of previous days' values). Note in particular that `TempModel8` contains two sets of covariates, one for the mean and one for the variance (see equations (3), (6) and (7)).

You should also generate some of the residual plots for `IntModel5` and `TempModel8`, partly to reassure yourselves that the models provide a reasonable fit but also to give you an idea of what still needs to be done. In particular, notice the time trends in some of the annual mean residuals — hopefully these can be linked to changes in large-scale atmospheric predictors. Notice also the seasonal structure in the standard deviations of the precipitation intensity residuals — this is potentially problematic, but Rglimclim (like many other stochastic downscaling methods) currently is unable to resolve it.

There is one residual plot that we did not see when building the precipitation occurrence model, because it is only appropriate for continuous variables. It is a quantile-quantile plot for checking distributional assumptions:

```
par(mfrow=c(1,2),mar=c(4,3,4,2),mgp=c(2,0.75,0),lwd=2)
plot(IntModel5,which.plots=4,plot.cols=c("blue","black"),titles=FALSE)
title(main="Q-Q plot of standardised residuals\n(precipitation intensity model)")
plot(TempModel8,which.plots=4,plot.cols=c("blue","black"),titles=FALSE)
```

```
title(main="Q-Q plot of standardised residuals\n(temperature model)")
```

Notice the use of `titles=FALSE` in the `plot` commands: this suppresses the default plot titles, allowing the user to customise them if desired. The plot for the precipitation intensity model shows that the gamma distributions provide an an astonishingly good fit to the standardised residuals; the plot for the temperature model shows some lack of fit in the lower tail, but otherwise seems reasonable.

2.6 Building a bivariate model

Having built multi-site models for precipitation and temperature separately, our next task is to link them to produce a bivariate model that (hopefully) can reproduce the dependence between the two variables. In Rglimclim, this is done by including *one* of the variables as a covariate in the model(s) for the other. The justification for this is that if $\mathbf{R}_t$ and $\mathbf{T}_t$ denote vectors containing respectively the precipitation and temperature values for all sites on day t, then the joint density (pdf) of $\mathbf{R}_t$ and $\mathbf{T}_t$ can be factorised as

$$f_{\mathbf{R},\mathbf{T}}(\mathbf{r},\mathbf{t}) = f_{\mathbf{R}}(\mathbf{r})\, f_{\mathbf{T}|\mathbf{R}}(\mathbf{t}|\mathbf{R}=\mathbf{r}) \quad \text{or as} \quad f_{\mathbf{R},\mathbf{T}}(\mathbf{r},\mathbf{t}) = f_{\mathbf{T}}(\mathbf{t})\, f_{\mathbf{R}|\mathbf{T}}(\mathbf{r}|\mathbf{T}=\mathbf{t}) \ , \quad (9)$$

where the vertical bars denote conditional distributions. Our existing models for precipitation and temperature in fact define the marginal distributions $f_{\mathbf{R}}(\mathbf{r})$ and $f_{\mathbf{T}}(\mathbf{t})$ respectively — or, rather, they serve as *approximations* to these marginal distributions: if we can model one of the conditional distributions therefore (e.g. via a GLM), we have a fully bivariate model.

The next, question, then, is: which way round should we do it? Should we treat precipitation as the "primary" variable and extend the temperature model to include (functions of) precipitation as a covariate, or should we treat temperature as the "primary" variable and extend the precipitation models? In almost all of the literature on weather generation, precipitation is treated as the primary variable; but this is almost certainly because when weather generators were first developed in the early 1980s people knew how to regress temperature on precipitation (because it has something like a normal distribution) but they did not know how to regress precipitation on temperature. GLMs remove this constraint, but in doing so they force us to confront the question.

You might think that the question is academic, because both equalities in (9) are exact and lead to the same joint distribution. This would be true if our models were perfect; but they are not. As noted above, they are approximations, and it is possible that the approximation error will be smaller (and the weather generator performance correspondingly better) in one case than the other. Unfortunately therefore, we must think about it. Relevant considerations include:

- Are there physical considerations that suggest that one variable acts as a (direct or indirect) influence on the other? If so, the influencing variable should be considered as the primary variable, because statistical models are invariably more robust if their structure reflects the mechanisms governing the phenomena that they represent. In this particular instance, we could argue:

  - that precipitation influences temperature indirectly, particularly in winter, because winter precipitation is associated with extensive cloud cover and warmer temperatures; or
  - that temperature influences precipitation directly in summer, because summer precipitation in this region is dominated by convective events and convection is enhanced on warm days.

  This does not help.[8]

- Are there differences in data availability for the two variables? If one variable has many more observations than the other, then it may be pragmatic to treat this as the primary variable — because when we build a GLM we must (usually) discard any observation for which the covariate values are unavailable. If we treat the 'data-poor' variable as the primary variable therefore, we must subsequently discard many observations from the 'data-rich' variable, leading to an overall loss of precision.[9]

  Sometimes, this principle *does* help. In the current example, however, it doesn't: both temperature and precipitation have around 200,000 observations available.

2.6.1 Your task

To resolve this question, we are going to carry out a scientific experiment. Work in groups of 2–4 people each. Half of the groups will use temperature as the primary variable, and the remainder will use precipitation. Ultimately, we will compare the models in terms of their validation performance.

If you take temperature as the primary variable, then your task is to modify the precipitation occurrence and intensity models to incorporate (a function of) temperature as an additional covariate. Conversely, if you take precipitation as the primary variable, your task is to modify the temperature model (possibly in both the mean and variance components) to incorporate (functions of) precipitation as additional components.

In all cases, the first step is to add an extra 'component 4' line to the model definition, with the following generic format:

```
4    0.0000    0   XX    Y
```

where `XX` defines the covariate transformation (set it to zero for an untransformed covariate) and `Y` is either 1 or 2 depending on whether the covariate is precipitation or temperature. Note that the first 'code' is zero, indicating a covariate value on the same day as the predictand. If your chosen transformation requires additional parameters (e.g. an exponential decay rate) then, of course, you will also need to update the 'nonlinear parameters' section of the model definition in order to define this parameter.

[8] I have had many conversations with climate scientists on this topic. We have never got further than this!

[9] In Rglimclim, the `allow.incomplete.averages` argument to the `GLCfit()` command offers one possible solution to this problem, at least when the covariate enters as a (possibly weighted) average over all sites. See `help(GLCfit)` for details.

Here are some hints to help you with the task (see also Section 4 of the manual):

- Add the extra component *after* all of the other 'component 4' lines in the model definition. This way, you won't accidentally change the numbering of the other covariates (which could cause problems with interactions and nonlinear parameters).
- Use what you have learned about Rglimclim to inform your modelling: use the model outputs, diagnostic information, formal model comparisons and your scientific intuition. You may find it particularly helpful to review the way in which we identified an appropriate choice of structure for the previous day's covariate in the precipitation occurrence model: the present task is very similar.
- Think about the processes that are responsible for precipitation-temperature associations. These may help you to identify plausible transformations in your modelling. They may also suggest the inclusion of interaction terms: for example, the association between temperature and precipitation tends to be positive in winter (clear skies imply low temperature and low precipitation) but possibly negative in summer.
- Be structured and methodical! Don't be afraid to try several models; but *do* approach them in a systematic way, and **make sure that you name the models, and their definition files, in a way that will help you to follow the structure later**.
- Keep things as simple as possible. Thus, if you want to represent a seasonally varying effect, just work with the main sine and cosine components; similarly, interactions with site effects should be kept fairly simple. Interactions involving previous days' values should be treated with *extreme* care, particularly if the interacting variable shows strong time trends — such interactions can cause serious problems when simulating from the fitted model, because they can take the coefficients outside the range within which the model is stochastically stable.
- If you are taking temperature as a primary variable, you must consider its potential influence upon both precipitation occurrence and intensity. You therefore need to know how to modify the commands you have learned so far, to fit a precipitation intensity model. This is easy: just use `model.type="gamma"` instead of `model.type="logistic"` in any `read.modeldef` command.
- Similarly, if you are taking precipitation as a primary variable then you must consider its potential influence on both the mean and variance of the temperature distribution. This is slightly more difficult, because the temperature model requires *two* model definition files (one each for the mean and variance). If you wanted to generate some definition files from the existing `TempModel8`, you could go

  ```
  write.modeldef(TempModel8,c("TempModel9_MeanInit.def","TempModel9_Varinit.def"))
  ```

  Then after editing the mean and / or variance definition files, you must call `read.modeldef` *twice*, each time using `model.type="normal-heteroscedastic"` and specifying `which.part="mean"` or `which.part="dispersion"` as appropriate. For example:

```
TempModel9_Var.init <- read.modeldef("TempModel9_VarInit.def",
                                     model.type="normal-heteroscedastic",
                                     which.part="dispersion",siteinfo=siteinfo,
                                     var.names=c("Precipitation","Temperature"))
```

Then, to fit the model, you must modify the call to `GLCfit()` slightly, for example

```
TempModel9 <- GLCfit(TempModel9_Mean.init,dispersion.def=TempModel9_Var.init,
                     which.response=2,siteinfo=siteinfo,
                     data.file="NorthIberiaPrecipTemp_1960-1990.dat",
                     diagnostics=1,nprev.required=-1,
                     cor.file="TempModel_Corr.dat")
```

Notice here:

- The use of `dispersion.def=TempModel9_Var.init` to specify the structure of the variance model
- The use of `which.response=2`, to specify that we are modelling the second variable in the data file (this argument was not needed when modelling precipitation occurrence, because by default Rglimclim assumes that we are modelling the first variable in the file).

These normal-heteroscedastic models take much longer to fit than the precipitation models. This is because the algorithm iterates between fitting a mean model with weights determined by the variance model, and fitting a variance model to residuals from the mean model. For details, see Appendix C.1.2 of the manual.

Good luck!

2.7 Incorporating atmospheric predictors

You have now built a bivariate, multi-site weather generator. The only thing missing is some way of including a climate change signal — we are not downscaling yet! For this final step, we must include additional covariates representing large-scale atmospheric structure. In Rglimclim, these are referred to as *external* covariates and are provided in separate data files.

Recall from Section 1.1.3 that there are two sets of atmospheric predictor data available to us: a set of circulation indices, and a set of weather classification information. These represent two rather different approaches to downscaling. Again, we will use this workshop to compare them: half of the groups will use the circulation indices as atmospheric predictors, and the remainder will use the weather classifications. The subsequent validation will then tell us which approach is the most effective for this region.

Daily time series for all of the predictors are provided in file `DailyPredictors.dat`. Click on the filename to open it in RStudio (it will issue a warning message about the file size, but continue anyway). Scroll down to lines 42–53 of the file, where you can

see that it contains data on 11 variables: the first six are the circulation indices ('Z8' in the descriptions refers to region Z8 from **?**) and the remaining five define the weather classification information. A few lines further down, you see

```
************************END OF PREDICTOR DEFINITION************************
19570901   16.4294  -1.1436  ... 0.0000   0.0000   1.0000   0.0000   0.0000
```

This first row of data is for 1st September 1957. The first six fields after the date represent the the values of the circulation indices, and the remaining fields encode the weather classification: the value of 1 in the ninth field tells us that this day belongs to the "S circulation" weather state. This use of 0/1 variables to encode categorical data is common in statistical modelling.

There is also a file `MonthlyPredictors.dat`, which contains monthly averages of the values in `DailyPredictors.dat` (note that monthly averages of the 0/1 weather state variables are the proportions of days in each state, and can be interpreted as 'circulation tendencies' for each month). In real downscaling applications, it is often helpful to work with monthly rather than daily predictors — apart from anything else because climate models do not agree on the number of days in a year, but they *do* agree that there are 12 months. However, we will just focus on the daily predictors in these sessions.

2.7.1 Your task

You must now update your bivariate model to incorporate EITHER the circulation indices OR the weather state indicators (but not both). You need to decide which variables to include, and in which models (precipitation occurrence, precipitation intensity, temperature mean, temperature variance). The predictors are all at a daily scale, so once again they can be defined to Rglimclim by adding extra 'component 4' lines to the model definition, along with appropriate interaction terms if necessary. On page 16 of the manual, Table 2 tells us that codes of 51 upwards correspond to 'external' daily predictors. So, to add all of the circulation indices to a model, just include the lines

```
4    0.0000   51
4    0.0000   52
4    0.0000   53
4    0.0000   54
4    0.0000   55
4    0.0000   56
```

to the corresponding definition file (do not attempt to experiment with lagged values of the predictors, this will make your life too difficult!).

If you are using weather states instead of circulation indices, you might think that you should include lines

```
4    0.0000   57
4    0.0000   58
4    0.0000   59
```

```
4    0.0000   60
4    0.0000   61
```

to the definition file. This will produce an error when you try and fit the model, however. The reason is that the weather state variables are perfectly linearly dependent: they always sum to 1, and therefore you don't need all of them. You must therefore choose one of the states as a 'reference state', and omit it from the model.

A suggested strategy for adding these 'external' predictors to your models is: start by adding all of them, together with any interactions that you think could be important / useful. Fit the resulting model, and examine the output to see if there are any terms, or groups of terms, that appear insignificant. If so, delete them (make sure that you correct any changes to the covariate numbering, in the 'interaction' and 'nonlinear parameters' sections of the definition files), refit the model and then use the `anova()` command to check that the deletions were justified.

When you have finished, save your models to the file `IberiaWG.rda` file using the `save()` command, for example

```
save(OccModel8,IntModel5,TempModel8,file="IberiaWG.rda")
```

3 Session 2: testing the generator

In this second practical session, we will use our weather generators to simulate multisite daily sequences of temperature and precipitation, and we will assess their performance.

You might think that we assessed the model performance already, by considering residual plots and other diagnostics. However, those diagnostics were for checking the distributions of each variable *conditional on all of the other covariates* in the models — in particular, the previous days' weather variables and the atmospheric predictors, which change from day to day. In applications, we are usually interested in the overall, *marginal* properties of the variables. For weather generators of realistic complexity, it is not possible to deduce the marginal properties so we must use simulation instead. Some of the smaller model deficiencies may become more important when simulating; conversely, some of the larger errors (such as the failure of the models to capture the full range of seasonality in the variance of precipitation intensity) may be less serious in the marginal distributions.

Once again, there is an R script for use in this session (`ModelTesting.r`), in your `PRACTICALS/` directory. Start up RStudio, change to the `PRACTICALS/` directory and open this script. Run the first two commands, which load Rglimclim along with the file `IberiaWG.rda` that you saved at the end of the first session. The remaining commands in the script use the univariate climatological weather generators `OccModel8`, `IntModel5` and `TempModel5`. As you work, adapt the script to use your own models.

3.1 Introduction to simulation

We will start by producing 20 simulations for the 10-year period 1970–1979, at all of the sites in the study area — or, more precisely, at all of the sites defined in the `siteinfo` object. Obviously, data from this period were used to fit the models, so this is not a 'genuine' test of their performance. However, if this initial simulation exercise reveals any serious problems, we have a chance to refine the models before attempting a validation using data from a period (1991–2002) that was not used to fit the models.[10]

Simulation is done using the `GLCsim()` command:

```
set.seed(2000)
sim1970to1979 <- GLCsim(list(list(Occurrence=OccModel8,Intensity=IntModel5),
                                    TempModel8),
                        nsims=20,start=197001,end=197912,impute.until=196912,
                        which.regions=0:1,simdir="./SimFiles",
                        file.prefix="Sim1970-1979")
```

The first of these commands the random number generator to a repeatable initial state, so that the results can be reproduced exactly.[11] The second carries out a simulation and stores the information in an object called `sim1970to1979`. Note the following:

- The first argument to `GLCsim()` is a `list` containing two objects, each defining a model for a single variable (precipitation and temperature respectively). The first object, defining the precipitation model, is itself a list (!) containing the occurrence and intensity models, named explicitly as `Occurrence` and `Intensity`. This naming is important: without it, `GLCsim` would try to perform a multivariate simulation in which one variable was modelled using `OccModel8` and the other using `IntModel5`.
- The next three arguments `nsims`, `start` and `end` are self-explanatory. Start and end dates are given in the form `YYYYMM` where `YYYY` is the year and `MM` is the month. The simulation starts on the first day of `start` and ends on the last day of `end`.
- The argument `impute.until` is used to control the imputation behaviour of the routine. This is discussed in more detail below.
- The routine will optionally generate an output file for each simulation, containing monthly series of average precipitation and temperature for a selection of subregions that are defined within the site information databases. The `which.regions` argument controls this selection. By default, the routine generates these monthly series only for region 0 (i.e. the entire area); here we request monthly series for subregion 1 as well (you may remember that this was defined to include all of the 'official' VALUE sites, early in the first session).

[10] In real applications we would probably produce more simulations for the entire 1960–1990 calibration period. However, the output files are large, and the simulation process is not instantaneous, so we will start with a small number of short simulations.

[11] Actually, there is a small bug in Rglimclim which means that you only get *exactly* the same results if you restart R and run the same commands again. I know, this is annoying, but I haven't found a way of fixing it (it relates to the interface between R and the underlying Fortran code).

- The routine will also generate a daily output file for each simulation, in the same format as the original data file (`NorthIberiaPrecipTemp_1960-1990.dat`) that has been used throughout the modelling process. Together with the monthly output files therefore, this simulation will generate 40 output files in total. They will be stored in the subdirectory defined by the `simdir` argument, and the output file names will be generated automatically. The `file.prefix` argument allows the user to specify a prefix for each output file name; we'll see how this is used, below.

Note that the `sim1970to1979` object does *not* contain the simulated data: it merely contains information about what was simulated and where the data are stored. To see this, type `sim1970to1979`:

```
Object of class GLCsim:
=======================

Variables taken from data file NorthIberiaPrecipTemp_1960-1990.dat
Variables simulated:
1. Precipitation (model type: logistic-gamma)
2. Temperature (model type: normal-heteroscedastic)

Simulation period: 1/1970 to 12/1979
No imputation performed
20 realisations generated

Output files generated: daily and monthly
Daily output written from 1/1970 to 12/1979
Monthly summaries written for the following regions:
0 Northern Iberia
1 VALUE stations
Output directory: C:/[output truncated]/PRACTICALS/SimFiles
Prefix for output filenames: Sim1970-1979
```

Next, use command `list.files("SimFiles/")` to inspect the output files that have been generated. Notice how the file names have been automatically generated, using the `file.prefix` argument supplied to the `GLCsim()` routine.

Unfortunately, the daily simulation files are too large for the RStudio editor. However, we can look at the first few lines of one of them:

```
read.fwf("SimFiles/Sim1970-1979_Daily_Sim01.dat",widths=c(4,2,2,4,6,6),n=10)
```

The `read.fwf()` command reads files with fixed column widths. It is very slow when you need to read large files (if you need to read the entire file, use `read.GLCdata()`), but it is useful when you only want a few lines. The variables are year, month, day, 4-character site code, precipitation and temperature. You will all get different results (presumably) because you are all using different models — and perhaps different random number seeds.

Now open the monthly file `SimFiles/Sim1970-1979_Monthly_Sim01.dat` (this *will* open in the RStudio editor). Each line contains 28 values: the first two are the year

and the region code (note that 1970 appears twice, once for region 0 — representing an average over all of the stations in `siteinfo` — and once for region 1 — representing an average over the VALUE stations). The remaining columns are in groups of 13. Each group represents one variable, giving 12 monthly means and an annual mean.

The monthly output files are small and easy to read. Potentially, they allow impacts modellers to identify simulation periods that might be particularly interesting. For example, a hydrological model might be computationally intensive to run: a hydrologist may therefore wish to select 'wet' or 'dry' simulation periods to explore the range of hydrological responses that might be expected. The monthly files provide a convenient way of identifying such periods in the simulation output.

Close the monthly output file before proceeding.

3.1.1 Multiple imputation

The daily (and possibly monthly) simulation files can now be used for input into impacts models. However, we should first assess the credibility of the simulations. This can be done by comparing their properties with those of observations. However, the observations are incomplete: there are some missing values, so the 'observed' properties are subject to uncertainty. This can be assessed by generating additional simulated sequences in which the missing values are sampled from their conditional distributions given the available observations, and by calculating properties of interest for each of these additional sequences. The procedure is known as *multiple imputation*, and the `GLCsim()` routine performs it automatically unless prevented from doing so by the `impute.until` argument. So, to produce 3 imputations of the missing observations in our data set:

```
obs1970to1979 <- GLCsim(list(list(Occurrence=OccModel8,Intensity=IntModel5),
                             TempModel8),
                        nsims=3,start=197001,end=197912,
                        which.regions=0:1,simdir="./SimFiles",
                        file.prefix="Obs1970-1979")
```

The `GLCsim()` call is exactly the same as before, except that:

- The `impute.until` argument is omitted so that the software will condition on all available observations throughout the simulation period (note that in the earlier call, `impute.until` defined a date prior to the start of the simulation, in order to prevent the routine from doing imputation).
- The number of realisations is 3 — again, this is just for illustrative purposes. The appropriate number of imputations is discussed in more detail below.
- The `file.prefix` is now `Obs1970-1979`, so that we can distinguish the files containing simulations from those containing imputations.

We can now assess the simulation performance by calculating summary statistics for each simulation, and comparing the distribution of these summary statistics with the

corresponding observed value (or imputation range). For example, we might be interested in the mean precipitation for January. Each of the 20 simulations will produce a different January mean precipitation, so we have a simulated distribution for this quantity. The test of simulation performance is whether or not the *observed* mean precipitation can plausibly be considered to belong to this distribution.

Rglimclim offers a range of plots to help assess simulation performance. To access these, some preprocessing is needed. We will calculate some summaries for two of the VALUE sites, and also for the average of all the VALUE sites (for the full range of options available, (type `help(summary.GLCsim)`). Here are the preprocessing commands:

```
seasons <- list(3:5,6:8,9:11,c(12,1,2))
sim.summary <- summary(sim1970to1979,season.defs=seasons,
                       thresholds=c(0,NA),
                       which.sites=c("1394","3910"),which.regions=1)
obs.summary <- summary(obs1970to1979,season.defs=seasons,
                       thresholds=c(0,NA),
                       which.sites=c("1394","3910"),which.regions=1)
```

To see what has been done here, type `sim.summary`. The system reports the variables, sites and regions for which summaries have been calculated, and also which summary statistics are available. Notice the use of the `thresholds` argument in the `summary` commands above: if this is supplied, the system will compute the proportion of threshold exceedances (corresponding to the proportion of wet days when, as here, a zero threshold is set for precipitation), along with the mean and standard deviation of these exceedances. The `NA` threshold for the second variable means that threshold exceedance statistics are not calculated for temperature. Notice also the use of the `season.defs` argument: this will compute annual time series of seasonal means for user-defined 'seasons' (which are just groups of months, specified here in the `seasons` object).

There is a lot of information in the summary object. Some plots would be helpful; these can be produced using the `plot` command. This is very flexible (see `help(summary.GLCsim)`). It is helpful to extract the names of the summary statistics, in a format that can be used directly for input into this `plot` command:

```
stat.names <- dimnames(sim.summary$Daily$Regions)$Statistic
```

Look at the `stat.names` object: the first 10 statistics relate to the marginal distributions of each variable, and the remainder are inter-variable correlations. Start by producing some plots for marginal precipitation statistics:

```
if (dev.cur()==1) x11(width=8,height=6)
par(mfrow=c(2,5))
plot(sim.summary,imputation=obs.summary,which.sites=NULL,
     which.timescales="daily",which.variables="Precipitation",
     which.stats=stat.names[1:10])
```

This produces 10 plots, each showing the simulated distributions of a different summary statistic for each month of the year. The grey bands indicate the percentiles of the simulated distributions, while the black bands show the range of values obtained from the 3

imputations (note that the `imputation` argument is optional — if we were downscaling future climate projections for example, we would not have any observations / imputations to add to the plot). The plots are too small to accommodate their titles here; for finer control, and the selection of which plots are produced, see the subsequent examples and the help page. The summaries produced are the mean; standard deviation; maximum and minimum values (the minimum being zero for every month of every simulation, unsurprisingly); autocorrelation at lags of 1, 2 and 3 days; proportion of wet days; and the mean and standard deviation on wet days only (defined as exceedances of the zero threshold).

If the simulated time series are realistic, the observed or imputed values for each statistic should look like a sample from the simulated distributions. Informally, this means that the imputation envelopes should lie mostly within the simulated distributions and, moreover, should traverse the range of those distributions (i.e. there should be some values at the lower end, some at the upper end and some in the middle — although, with only 20 simulations here, you should not be surprised if the observations occasionally fall outside the simulation range). The imputation range indicates how much uncertainty is associated with missing data values.

A similar plot can be produced for temperature. We didn't define a threshold for temperature in the `summary` command, so we cannot consider threshold exceedances here. We can add some colour, however:

```
par(mfrow=c(2,4))
plot(sim.summary,imputation=obs.summary,which.sites=NULL,
     which.timescales="daily",which.variables="Temperature",
     which.stats=stat.names[1:7],colours.sim="colour")
```

You may feel that this default colour scale is not very intuitive for temperature (it was designed with precipitation in mind). We will fix this later. Next though, we can look at inter-variable correlations:

```
par(mfrow=c(2,2))
plot(sim.summary,imputation=obs.summary,which.sites=NULL,
     which.timescales="daily",which.stats=stat.names[11:12],
     colours.sim="colour")
```

You can read the plot titles now: they show that these plots are for the daily time series of mean precipitation and temperature, averaged over all of the VALUE stations. You may want instead to look at the behaviour for a specific site:

```
plot(sim.summary,imputation=obs.summary,which.sites="1394",
     which.regions=NULL,which.timescales="daily",
     which.stats=stat.names[11:12],colours.sim="colour")
```

The imputation range is very narrow here: site 1394 has a complete record (this is why it was chosen for use in the VALUE experiment).

In some applications, it is important to reproduce the variation in means or totals over monthly or longer time scales. In the `summary` commands above, we calculated summaries for four 3-month seasons. To visualise these we can use the `plot()` command again, but with `which.timescales="monthly"`:

```
quantiles <- c(0,0.1,0.25,0.5,0.75,0.9,1)
plot(sim.summary,imputation=obs.summary,which.sites=NULL,
     which.timescales="monthly",which.variable="Temperature",
     quantiles=quantiles,colours.sim=heat.colors(length(quantiles)-1),
     ylabs=rep(expression(degree*C),4))
```

This example illustrates some further options in the `plot` command, giving the user detailed control over the output.

3.1.2 Connection with VALUE measures

The summary statistics produced by Rglimclim are intended to help you understand how well your weather generator performs with respect to the marginal distributions of each variable, the dependence between variables and the time series properties such as seasonality, autocorrelation and interannual variability. In VALUE, such summary statistics are referred to as 'indices' to quantify different aspects of the weather. The full list of VALUE-endorsed indices is at http://www.value-cost.eu/indices: Rglimclim computes many, but not all of them. For example, the representation of inter-site dependence is not explicitly addressed at present. It can be assessed indirectly, however, via the regional mean summary statistics — because, for example, the variability of a 10-site average depends both on the variation in the individual series and on the correlation between the sites.

As well as defining indices, VALUE defines measures for assessing how well they are reproduced. Rglimclim does not attempt to calculate such performance measures: the plots allow an informal assessment of performance. If a user wishes to calculate such measures however, they can access the calculated indices in the `summary` objects (e.g. the simulated mean for simulation i, month j, site k and variable ℓ is contained in the $(i, j, k, \ell, 1)$ element of the `Daily$Sites` component of such an object). Moreover, users can write routines to examine alternative indices that are not provided directly by Rglimclim. For example, the sample script provides code to compare the simulated distribution of annual precipitation maxima with the distribution obtained by fitting a Generalised Extreme Value (GEV) distribution to the observed annual maxima. This allows you to assess the weather generator performance with respect to rare events. Don't worry too much about the details of these commands — you can study them in your own time later. But *do* run the commands, to find out how the quantiles of your simulated annual maximum distribution compare with those of the observations, both for a single site and for the regional mean precipitation. Are your simulated distributions of annual maxima consistent with what you would expect from the GEV return level plots?

The results above may have revealed some problems with your model structure. If you want to adjust your models slightly therefore, do so before moving on to the next section.

3.2 Out-of-sample validation

Your final task is to assess the performance of your weather generator for a period (1991–2002) that was not used to develop the models. We will do this properly, using 100

simulations and 39 imputations.[12] The commands are:

```
sim1991to2002 <- GLCsim(list(list(Occurrence=OccModel8,Intensity=IntModel5),
                             TempModel8),
                        nsims=100,start=199006,end=200208,impute.until=199005,
                        which.regions=0:1,daily.start=199101,
                        simdir="./SimFiles",file.prefix="Sim1991-2002")
obs1991to2002 <- GLCsim(list(list(Occurrence=OccModel8,Intensity=IntModel5),
                             TempModel8),
                        nsims=39,start=199006,end=200208,
                        which.regions=0:1,daily.start=199101,
                        data.file="NorthIberiaPrecipTemp-1990-2012.dat",
                        simdir="./SimFiles",file.prefix="Obs1991-2002")
```

While these commands are running (probably 7–8 minutes in total), note the following:

- Both sets of simulations start in June 1990 instead of January 1991. This is because the models include lagged values of both variables; to initialise a simulation it is therefore necessary to provide some initial values. Where possible, Rglimclim initialises its simulations with data immediately prior to the simulation period. If we started the simulations in January 1991 therefore, they would all be initialised using observations from the end of December 1990: thus they would be very similar to each other for the first part of January 1991. By starting the simulations six months earlier, the initialisation will have negligible effect on the period of interest. This is directly analogous to the 'spin-up' period for a numerical climate model.
- Although the simulations start in June 1990, we are not interested in the output until January 1991. The argument `daily.start=199101` ensures that the output files only contain data for the period of interest.
- When fitting models, we used a data file containing data from 1960–1990. In the second command above, the argument `data.file="NorthIberiaPrecipTemp-1990-2012.dat"` is needed to define the data used for imputation from 1991–2002.
- The simulation period ends in August 2002, not December 2002. This is because the ERA40 predictor data only run to August 2002.

3.2.1 Your task

You should now examine a variety of plots to assess the performance of your weather generators. You should aim to answer the following questions:

- How well are the marginal aspects reproduced for each variable? Are any aspects reproduced particularly poorly? (for example, from the model-building diagnostics, you might expect precipitation variability to be reproduced poorly in summer).

[12] 39 seems like a strange number. It is chosen because the resulting range of values for each summary statistic forms a 95% prediction interval for the value that would have been obtained from complete data (**proof**: exercise!).

- How well are the temporal aspects reproduced? These include features such as seasonality as well as autocorrelation.
- How well are spatial aspects reproduced? (look at summary statistics for the VALUE regional mean time series).
- Based on your results, and your experience while building the models, what would be your advice to a user who is considering using your weather generator in an assessment of climate change impacts?

We will aim to finish the session with a discussion: each group will present their results briefly, summarising their model structure and atmospheric predictors, along with the simulation performance. Hopefully this will provide some insight into questions such as: is it better to model precipitation conditional on temperature, or vice versa? Is a weather classification approach better or worse than an approach based directly on circulation indices? What are the main strengths and weaknesses of this GLM-based approach to weather generation?

4 Finally . . .

These practical sessions have introduced a lot of material. Don't worry if you didn't understand all of it. I hope that some of you will want to explore these ideas further in your own work; and that this document is a useful guide for you.

The latest version of Rglimclim is always available from http://www.homepages.ucl.ac.uk/~ucakarc/work/glimclim.html. There is a mailing list, used to notify users of updates. If you would like to receive this information, please send me an email and I will add you to the list. Also, if you would like to see some features that are not currently available then please let me know by email. I will add them to my "to-do" list (which is, however, very long!).

Thank you for your attention and your patience.

Richard Chandler
London, 30th October 2014