RGLIMCLIM

A multisite, multivariate daily weather generator based on Generalized Linear Models

User Guide

Richard Chandler
Department of Statistical Science
University College London
Gower Street
London WC1E 6BT
ENGLAND
r.chandler@ucl.ac.uk

October 19, 2013

Contents

1 Introduction

This software is designed for the modelling and simulation of univariate or multivariate daily weather sequences at single or multiple sites. It evolved from the earlier Glimclim package, which was a suite of FORTRAN programs that were developed from the mid-1990s onwards for the modelling and simulation of daily rainfall sequences. The main differences between Rglimclim and the original FORTRAN package are:

- Rglimclim has been ported to the R programming environment (R Core Team, 2013) which makes for a much more convenient user interface. Most of the computations are still done using FORTRAN code, however: thus the package combines the computational efficiency of FORTRAN with the convenience of R.
- Rglimclim has multivariate simulation capability: it can generate daily series of multiple variables at multiple sites, preserving inter-site and inter-variable relationships.
- Rglimclim adds new model classes, based on normal distributions, to the gamma and logistic regression classes that were available in Glimclim.
- The R interface makes it much easier to save and update models, and to transfer and visualise information.

For references describing the methods used in this software, see Wheater et al. (2000), Chandler and Wheater (2002), Yan et al. (2002), Chandler (2005) and Yan et al. (2005) for example. The Appendix to this manual gives some technical details, for the interested user.

A basic level of IT competence is assumed throughout — in particular, the ability to edit text files. Some familiarity with R is also assumed.

A list of known bugs and problems, with suggested workarounds, is given at the end of this manual.

2 Installing the software

Rglimclim is regularly updated, in response to personal research requirements and to user requests. The latest version is always available from `www.homepages.ucl.ac.uk/~ucakarc/work/glimclim.html`. To receive notifications of updates, please email the package author (`r.chandler@ucl.ac.uk`).

To use the software, a working R installation is required (preferably version 3.0 or above). R can be obtained from the CRAN link at `www.R-project.org`. It is a cross-platform environment: download and install the version appropriate for your operating system.

Once R is installed, the procedure for installing Rglimclim depends on your operating system. Instructions are given here for Windows and Unix systems.

2.1 Windows systems

For Windows users, a precompiled binary distribution is supplied as a zip archive named `Rglimclim_x.y-z.zip` where `x.y-z` indicates the version number. Download and save this from the Rglimclim web page (address given above). Next, start up R with administrative privileges (depending on your system configuration, you may need to right-click the R icon on your Windows desktop and select 'Run as Administrator'). Finally, from the `Packages` menu, select 'Install packages from local zip files', navigate to the downloaded file and select it.

The Windows distribution contains both 32- and 64-bit versions of the package; the appropriate one(s) will be selected automatically to match your R installation.

2.2 Unix systems

On Unix systems, the package must be compiled from its source code which is supplied as a tarball named `Rglimclim_x.y-z.tar.gz`. Download and save this from the Rglimclim web page (address given above). Next, open up a terminal and navigate to the directory where you saved the tarball. The package can now be installed from the Unix prompt using

```
R CMD INSTALL --html --clean Rglimclim
```

You will need administrative privileges for this. Precise details are implementation-dependent — use `sudo` on Ubuntu systems, for example. The command above will ensure that HTML help files are installed and that the installation cleans up after itself. To see the other installation options that are available, type

```
R CMD INSTALL --help
```

2.3 Other operating systems

Users of other operating systems should build the package from the source tarball `Rglimclim_x.y-z.tar.gz`. If unsure how to do this, read the R documentation (start up R, type `help.start()` and follow the 'R Installation and Administration' link).

3 Using the package

Once the package is installed, it can be loaded within R by typing

```
> require(Rglimclim)
```

```
Loading required package: Rglimclim
Use 'help("Rglimclim-package")' to get started
```

New users should follow the hint here:

```
> help("Rglimclim-package")
```

or

```
> help("Rglimclim-package",help_type="html")
```

if your R system is not set up to produce HTML help by default.

The main routines in the package are `GLCfit()` and `GLCsim()`, for fitting and simulating models respectively. The help pages for those routines give full details of their arguments and outputs. The purpose of this manual is to supplement the help pages, in particular by providing comprehensive documentation for the coding used in model definition files (see Section 3.8 below), by providing some hints on use in Section 4, by providing a simple worked example in Section 5 and by providing technical details of the underlying theory and algorithms in the Appendix. New users are advised to work through the example in Section 5 to familiarise themselves with the package. First however, it is helpful to set out some of the underlying rationale and principles.

Rglimclim is designed to allow flexibility in fitting fairly complicated generalized linear models (GLMs[1]) to daily climate/weather data, and to avoid some of the tedious manipulation of data files that would be necessary if standard software packages were used to perform this kind of analysis. It exploits the fact that all exercises involving fitting, and simulating, GLMs for daily weather series must inevitably share common features, as follows:

- Models are to be fitted to data from one or more sites, for which various attributes (e.g location) are known. Flexibility in the placement and known attributes of sites is achieved via the use of a database containing site information.
- Typically, possible covariates fall into a small number of categories, as follows:
  - A constant term.
  - Site effects (i.e. systematic spatial variation)
  - 'Year' effects (e.g. long-term trends).
  - 'Month' effects (e.g. seasonality).
  - 'Day' effects (e.g. day-to-day temporal autocorrelation).

[1]It is assumed that the reader has a basic working knowledge of GLMs. A brief summary of the main features is provided in Appendix A. Gentler introductions can be found, for example, in Krzanowski (1998), Davison (2003) and Chandler and Scott (2011, Chapter 3).

– Interactions.

The software exploits this small number of categories, and treats each separately. In each category a variety of choices can be made regarding model structure, by selecting from a 'menu'.

'External' effects (such as ENSO or the North Atlantic Oscillation) are dealt with under the appropriate timescale — for example, if you wanted to use a monthly ENSO index as a covariate in your model, this would be treated as a 'monthly' effect; if you wanted to use a 'winter NAO' series (one value per year), it would be a 'yearly' effect.

Climate datasets often have other unusual features relating to measurement methods. For example, in daily raingauge data any non-zero amount that is less than some small threshold may be recorded as a 'trace' amount, because it is too small to be measured accurately. Such features pose potential problems for statistical analysis. The software aims to provide methods for dealing with them.

The original Fortran version of Glimclim made extensive use of definition files to define sites and models. All of these had to be prepared manually, with a consequent risk of error. Rglimclim contains routines that will read old Glimclim definition files; however, it is intended that eventually the primary means of defining sites, models etc. will be via manipulation of R objects so that definition files will no longer be needed. At present however, some definition files are still necessary; and model definition files in particular make extensive use of coding as detailed below.[2] These files must be prepared manually (templates are provided with the distribution), with a consequent risk of error. Whenever a model definition file is read into the system therefore, it is always a good idea to `print` the resulting object (the system will produce an interpretable description of what has been defined) and check carefully that the definition is what was intended. Whenever a model has been fitted, the `write.modeldef()` routine in Rglimclim can be used to generate an updated definition file, which can subsequently be edited to make minor changes to a model. This feature takes some of the pain out of the process.

3.1 Distributions available

Most software packages for fitting GLMs (for example the `glm()` routine in R) offer a wide range of different distributions. The range available in Rglimclim is more limited, and reflects the kinds of situations that are most commonly encountered in climatological data. Although it is straightforward in principle to estimate coefficient vectors for additional distributions, it is often difficult to devise tractable and realistic models for inter-site dependence and imputation (for example, the Rglimclim inter-site dependence structures for logistic regression models are the result of several years' research).

[2] One or two codes have a different meaning in Rglimclim than in Glimclim; this was necessary to provide the necessary multivariate functionality in a compact manner. If the user provides one of these codes, by default the software will issue a warning that the meaning has changed.

The following distributions / models are available in Rglimclim:

Logistic: in a logistic regression model, the response variable (Y_{st} say, denoting the response at site s on day t) takes values 0 or 1. Let $p_{st} = P(Y_{st} = 1)$, and let $\mathbf{x}_{st}$ be a corresponding vector of covariates. Then the logistic regression model takes the form

$$\log \frac{p_{st}}{1 - p_{st}} = \mathbf{x}'_{st}\boldsymbol{\beta}$$

for a coefficient vector $\boldsymbol{\beta}$.

Logistic regression is often used to model the occurrence of rainfall. In Rglimclim, if required the `GLCfit()` routine will automatically dichotomize a variable into 'zero' and 'non-zero' values when fitting logistic regression models (the precise behaviour depends on the setting of the `response.check` argument — `help(GLCfit)` for more details).

Gamma: for a response variable Y_{st} taking non-negative values, a gamma GLM with log link states that conditional on a covariate vector $\mathbf{x}_{st}$, Y_{st} has a gamma distribution with mean

$$\mu_{st} = \exp\left(\mathbf{x}'_{st}\boldsymbol{\beta}\right)$$

and shape parameter ν (which does not vary with $\mathbf{x}_{st}$).

Such models are often useful for studying positive-valued variables such as precipitation intensity and wind speed. Sometimes (as with precipitation intensity, for example), gamma GLMs should be fitted only to the non-zero values in a data set; once again, this can be achieved in the `GLCfit()` routine via an appropriate setting of the `response.check` argument.

Normal: A normal GLM is just a standard linear regression model in which Y_{st} is drawn from a normal distribution with mean

$$\mu_{st} = \mathbf{x}'_{st}\boldsymbol{\beta}$$

and constant variance σ^2. Such models are rarely appropriate for daily climate time series.

Normal-heteroscedastic: the main problem with standard linear regression models, when studying variables such as temperature and pressure at a daily time scale, is that the assumption of constant variance is usually violated. As an alternative therefore, a *normal-heteroscedastic* model allows both the mean and variance of Y_{st} to depend on (possibly different) covariate vectors $\mathbf{x}_{st}$ and $\boldsymbol{\xi}_{st}$:

$$\begin{aligned} Y_{st} &\sim N(\mu_{st}, \sigma^2_{st}) \\ \text{with} \quad \mu_{st} &= \mathbf{x}'_{st}\boldsymbol{\beta} \qquad (1) \\ \text{and} \quad \log \sigma^2_{st} &= \xi'_{st}\boldsymbol{\gamma}\ , \qquad (2) \end{aligned}$$

so that now the model contains two coefficient vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$.

3.2 Inter-site dependence structures

When simulating time series at multiple spatial locations, it is necessary to preserve the dependence between them. Rglimclim allows the use of a variety of dependence structures. Most of these are correlation-based although the precise details are model-dependent, as follows:

Normal and normal-heteroscedastic: for these models, inter-site dependence is specified via correlations between standardised residuals: if $Y_{st} \sim N(\mu - st, \sigma_{st}^2)$ under the model then the corresponding standardised residual is $(Y_{st} - \mu_{st})/\sigma_{st}$.

Gamma: for gamma models, inter-site dependence is specified via correlations between Anscombe residuals: these are defined (see Appendix D) in such a way as to have a distribution that is very close to normal. The rationale for this is that the multivariate normal distribution is the only multivariate distribution whose dependence structure is completely characterised by correlations.

Logistic: For binary response variables modelled using logistic regression, correlation-based dependence structures are defined in terms of latent Gaussian variables: specifically, a standard normal random variable Z_{st} is associated with site s on day t, and Y_{st} is set to 1 if $Z_{st} > -\Phi^{-1}(p_{st})$ where $\Phi(\cdot)$ is the standard normal distribution function and $p_{st} = P(Y_{st} = 1)$. Correlation between the latent variables $\{Z_{st}\}$ therefore induces association between the $\{Y_{st}\}$.

There are two other spatial dependence structures that can be used for logistic regression models. The first postulates the existence of two unobserved 'weather states' which respectively increase or decrease the probability that $Y_{st} = 1$ at all sites simultaneously, in such a way as to maintain consistency with the probabilities obtained from the logistic regression model. The other is obtained by modelling the distribution of the sum $\sum_{s=1}^{S} Y_{st}$ where S is the total number of sites. Both of these dependence structures are designed for use in situations where inter-site dependence is very high and does not vary appreciably with inter-site distance — for example, when modelling the occurrence of rainfall in catchments that are small relative to the typical scale of weather systems. For full details of these schemes, see Appendix E.3.

3.3 Task list

The steps required to carry out an 'end-to-end' analysis using Rglimclim are as follows:

1. *Collate data*

   (a) Assemble daily data on the weather variable(s) of interest, along with data on any 'external' covariates that vary on daily, monthly or annual time scales.

These data must be provided in external files with tightly specified formats: the `write.GLCdata()` and `write.GLCexternal()` commands can be used to generate these files.

(b) Create an R object containing information about the locations from which daily data are available, or for which simulation is required. This can be done either by reading definition files using the `read.siteinfo()` routine, or from existing R objects using the `make.siteinfo()` routine.

Optionally, sites can be assigned to 'subregions' for which separate summaries can be calculated when simulating from the fitted models. The motivation for this comes from applications in hydrology, where it can be important to produce realistic weather sequences over subcatchments in order to reproduce correctly the river flow in a larger catchment. Subregions can be defined using the `define.regions()` routine or (for compatibility with old versions of Glimclim in which subregions were defined via definition files) via the `read.regiondef()` routine.

If subregions are defined, this must be done *before* calling `read.siteinfo()` or `make.siteinfo()`.

2. *Build models*

(a) Decide on an appropriate class of models to represent the variable of interest. This will usually be determined by the nature of the variable — for example, if it takes non-negative values then a gamma model is the natural choice from those available in Rglimclim, whereas if it takes both positive and negative values then either a normal or normal-heteroscedastic model will probably be more appropriate.

(b) Write a file to define the structure of a fairly simple model, use the `GLCfit()` routine to estimate its parameters, and write the result to a new definition file using `write.GLCmodeldef()`.

(c) Check the model: is there structure in the data that is not explained by the model (use the `plot()` and `summary()` methods for the fitted models to answer this question), or does the model contain terms that could be removed without degrading its performance (use the `print()` and `anova()` methods)?

(d) Amend or extend the model by editing the new definition file produced in step (b); refit, recheck and repeat until a reasonable model has been found.

In a multivariate setting (i.e. where more than one weather variable is to be modelled, preserving inter-variable relationships) this process should be carried out for each variable separately. Inter-variable relationships are accounted for by including (transformations of) variables that have already been modelled as covariates.

3. *Simulate from the fitted models*

(a) If necessary (for example if simulated data are required at different locations from those used for model calibration) generate a new database of site information

for the simulation using `read.siteinfo()` or `make.siteinfo()`, and assemble any necessary files of 'external' climate covariates needed to drive the simulations (e.g. using `write.GLCexternal()`). These might be the outputs from Atmosphere-Ocean General Circulation Models for example, in order to derive weather sequences corresponding to projections of future climate.

(b) Run the simulation routine `GLCsim()` to generate the required number of simulations for a specified time period and set of locations.

(c) To assess the credibility of the simulated sequences it is also helpful where possible to generate simulations in 'imputation mode' (discussed in more detail later in this manual) and to compare the properties of simulated and imputed sequences (the package provides `summary` and `plot` methods to enable this): this may reveal further deficiencies and hence lead to further model refinement.

The remainder of the section gives more details, where necessary, of inputs and model codings. New users should study this, in conjunction with the examples given in section 5.

3.4 The primary data file

Daily data on the weather variables of interest should be provided to the system in an ASCII file, the format of which is described in the help page for the `GLCfit()` routine under the `data.file` argument: at the R prompt, type

```
> help(GLCfit)
```

to view this. The `make.GLCdata()` command can be used to generate such a file from data that are already stored in a data frame within R; however, as noted in the `GLCfit` help, R can be slow when dealing with very large datasets.[3]

A potential limitation of the Rglimclim data file format is that only six positions are available for each record. This is a deliberate decision, made to try and avoid excessive file sizes. It does, however, mean that some variables must be transformed before they can be modelled using Rglimclim. For example, mean sea level pressure is usually recorded in millibars and typically takes values between 950mb and 1050mb. If it is recorded to two decimal places, then a value of (for example) 1012.23 occupies seven positions and hence cannot be used in an Rglimclim data file. To get round this, a user may choose either to round to one decimal place (on the grounds that pressure observations are unlikely to be accurate to two decimal places in any case)[4]; to divide all observations by 10 and write the results to two decimal places; or to subtract 1000 from each value. Modelling and simulation

[3] Rglimclim itself does *not* store much data internally in R objects: it merely passes the file locations to the underlying Fortran code. This ensures fast performance with large datasets.

[4] If this is done, the 'missing value' code in the data file must be changed from its default value of −99.99 to a number with one decimal place — this can be done using the `missval` argument to routines such as `GLCfit()` and `write.GLCdata()`.

can then be performed on the transformed values, and the simulated values transformed back to the original scale.

3.5 'External' data files

In many applications, it is of interest to examine the effects of 'external' covariates upon climate/weather variables. By 'external', we mean non-deterministic time-varying quantities (as distinct from trends, which can be regarded as deterministic) other than the variable under consideration. Examples include ENSO, sea surface temperature series and the North Atlantic Oscillation.

We classify such external covariates according to the timescale of available data (i.e. whether we have annual, monthly or daily time series). To simplify the structure of the input files, a separate ASCII file is used for each timescale. These files are only required if the user requests external covariates at the corresponding timescales.

The recommended way to generate files of external covariate data is to use the `write.GLCexternal()` routine. Because of this, the format of the files is not documented further here. However, the generated files contain header sections which give details of the file format: users wishing to know more should read these headers.

When generating external data files, it is the user's responsibility to ensure that the data are provided in in chronological order and with no missing dates (otherwise, the programs may not be able to find all of the required covariates). Missing data values should be coded as -9999.9 in the data files.[5]

When fitting models, cases with missing values of any covariates will be discarded from an analysis (so, your external covariate data are available for a different time period from your response variable, models will be fitted only to that period for which the records overlap).

The simulation routine `GLCsim()` will halt with an error message if you try to simulate over any period for which external covariate data are missing.

3.6 Defining subregions

In the original Fortran version of Glimclim, subregions were defined to the system using 'region definition files'. This is no longer necessary: if simulation summaries for subregions are required, simply create an R object of region definitions using the `define.regions()` routine[6] before defining the database of site information (see below). Alternatively, to read an old Glimclim region definition file, use the `read.regiondef()` routine.

Note that subregion information is provided to the Rglimclim system using R objects

[5] The `write.GLCexternal()` routine automatically converts `NA` values to -9999.9.

[6] This object is just a data frame containing two columns: `Region` (the region number, starting with zero for the whole area) and `Name` (the name of the subregion).

rather than via files: these objects are small, so it is easy to store them internally.

Subregion definitions are ignored when fitting models: they are used only when generating summaries of simulations produced using `GLCsim()`. There is no requirement to define any subregions: if none are defined, by default the system will proceed as though a single "whole-area" region has been defined.

3.7 Defining site information

As with subregion definitions, sites and their attributes are defined to the Rglimclim system via R objects, in this case of class `siteinfo`. The recommended way to create such an object is via the `make.siteinfo()` routine: at the R prompt, type

```
> help(make.siteinfo)
```

for more details.

Rglimclim uses site information in several ways: first to define *attributes* (for example geographical co-ordinates and site altitude) of each site, for use in the subsequent modelling; second to define descriptive text for the site names and attributes, from which to construct meaningful output labels; third to associate sites with subregions if required, as discussed above; fourth to define four-character identification codes for each site, that are used to link the site information with the primary data file (see the help for `GLCfit()`, under the `data.file` argument, for more details on this); and finally to provide information that can be used to make maps of the sites or to compute inter-site distances.

Any number of attributes can be defined for each site. By default, the system assumes that the first two attributes represent geographical co-ordinates to be used in plotting maps or computing inter-site distances; this behaviour can be changed if necessary using the `coord.cols` argument to `make.siteinfo()`. To see what attributes have been defined in a `siteinfo` object and in what order, just type the name of the object at the R prompt.

In the original Fortran version of Glimclim, site information was provided using site definition files. This is no longer necessary nor recommended, so the file format is not discussed here. The `read.siteinfo()` routine can be used, however, by users wishing to read old site definition files.

3.8 Model definition files

In Rglimclim, model definition files are used to define covariates for a model (along with their associated parameter values), and to select spatial dependence structures. When fitting models using `GLCfit()`, the parameter values in these files are treated as initial estimates for the numerical estimation algorithm. Fitted models are stored as R objects of class `GLC.modeldef`; the `write.modeldef()` routine can be used to generate updated definition

files while building models. When simulating however, the `GLC.modeldef` objects can be supplied directly to the `GLCsim()` routine without any further reference to model definition files (this is a difference from the original Fortran version of Glimclim, in which the simulation routines also read their model structures from definition files).

Specimen model definition files can be generated using the `write.modeldef()` routine in conjunction with some of the template model structures included with as part of the Rglimclim installation (see Section 5 below, and `help(GLCdemo)`, for more details of these templates).

Model definition files each contain a header which is 46 lines long, and contains details of the file structure. This header should not be altered. The line after this is reserved for future use. Line 48 is used to define a title, of up to 70 characters, for the model being defined. This in turn will be used by the system to label outputs.

The remainder of the file is used to define the model structure proper. Each row corresponds to a single parameter in the model, and contains five entries:

`COMPONENT VALUE CODE1 CODE2 CODE3 TEXT`.

The rows are read using the FORTRAN format `I5,F10.6,3I5,A40`. An explanation of the entries is as follows:

`COMPONENT` occupies the first 5 positions in the record, and is used to identify the type of quantity being defined. Valid entries are:

0 if the record relates to the constant term in the regression part of the model.

1 if the record relates to a site effect in the regression part of the model.

2 if the record relates to a 'year' effect in the regression part of the model.

3 if the record relates to a 'month' effect in the regression part of the model.

4 if the record relates to an 'day' effect, in the regression part of the model. This includes previous days' values, as well as any other covariate that varies on a daily timescale.

5 if the record relates to a 2-way interaction in the regression part of the model.

6 if the record relates to a 3-way interaction in the regression part of the model.

7 if the record relates to the nonlinear transformation of one of the covariates in the regression part of the model.

8 if the record relates to a global quantity such as a 'trace' threshold for raingauges. Such quantities are not strictly part of the model, but must be defined to the system somehow.

9 if the record relates to a dispersion parameter for the model (not used for logistic regression models, or for the components of models of type `normal-heteroscedastic`).

10 if the record relates to the spatial structure of the model.

Component	Code 1	Code 2	Code 3
0 (constant)	No codes used		
1 (site effects)	Number of attribute, in the order stored in the `siteinfo` object (see Section 3.7)	If present, label of nonlinear transformation (see Table 3)	Not used
2 (year effects)	**Up to 50**: Label of trend function (see Table 3) **51 upwards**: $(x-50)$th variable defined in file of 'external' annual data (x being the code entered).	Optional selection of lagged values of external covariate (if Code 1 >50).	Not used.
3 (month effects)	**1**: $\cos(2\pi \times \text{month}/12)$ **2**: $\sin(2\pi \times \text{month}/12)$ **3**: $\cos(2\pi \times \text{month}/6)$ **4**: $\sin(2\pi \times \text{month}/6)$ **11–22**: Individual month indicators (11 = Jan, 12 = Feb etc.) **51 upwards**: $(x-50)$th variable in file of 'external' monthly data.	E.g. to use covariate value 2 years/months ago, set this field to 2. To use next year's/month's value set to -1.	Not used
4 (day effects)	**See Table 2, page 16**		
5 (2-way interactions)	Indices of interacting main effects (first site effect is 1)		Not used
6 (3-way interactions)	Indices of interacting main effects (first site effect is 1)		
7 (parameters in nonlinear transformations)	Index of covariate for which transformation is being defined. See note 5, page 19.	Parameter being defined (1, 2 or 3 — see Tables 3 and 5)	**0**: treat parameter as known **1**: find ML estimate of parameter
8 (global quantities)	**1**: Threshold for defining 'small' positive values. See note 6, page 19.	Method for dealing with such values (See Table 6)	Not used
9 (dispersion parameter)	No codes required. NB logistic and normal-heteroscedastic models have no dispersion parameter. This field is ignored by model fitting programs.		
10 (spatial structure)	Label of spatial dependence structure used (see Table 7)	Number of parameter (see Table 7)	Not used

Table 1: Codes for specification of models in definition files. To be used in conjunction with Tables 2–7.

Code 1	Code 2	Code 3
0–10: value x days ago (see note 1 on page 19) **21**: $\cos(2\pi \times$ day of year$/366)$ (see note 3 on page 19) **22**: $\sin(2\pi \times$ day of year$/366)$ **23**: $\cos(2\pi \times$ day of year$/183)$ **24**: $\sin(2\pi \times$ day of year$/183)$ **31–42**: Smooth month adjustments (31 = Jan, 32 = Feb etc.). See note 4 on page 19. **51 upwards**: $(x - 50)$th variable defined in file dy_preds.dat.	Optional. If present and Code 1 $\leq$ 10, selects a transformation of previous days' values (see Tables 4 and 5). If Code 1 $>$ 50, selects lagged covariate values as in rows 2 and 3 of Table 1.	If present, refers to the number of a variable in a multivariate input data file, thus allowing for inter-variable dependencies in the modelling. If not present, the system will assume that the response variable is intended. **See note 2 on page 19.**

Table 2: Codes for specifying 'daily' effects in model definition files. This is row 4 of Table 1. See Tables 4 and 5 for further details on transformations and averaging.

The rows of a model definition file must be ordered according to the value of `COMPONENT`; if the rows are out of order, an error message will result.

`VALUE` is the value of the parameter being defined, occupying positions 6–15 of the record.

`CODE1`, `CODE2` and `CODE3` are used to define the precise details of the model to the system. Their interpretation depends on the value of `COMPONENT`. In general, it is not necesary to define all three codes. If they are all defined, `CODE1` occupies positions 16–20 of the record, `CODE2` occupies positions 21–25 and `CODE3` occupies positions 26–30. See Tables 1–7 for full details on coding.

`TEXT` contains descriptive text for this record, and appears after position 31. It is intended to help make the definition file readable by the user, and is not required or used by the program.

See Section 5 below for examples of model definition files.

3.8.1 Notes on Tables 1–7

1. In Table 2, when working with multivariate data a value of zero can be entered as Code 1. This defines a model in which one variable depends on the value of another for the same day. Obviously, in this case the variable being referenced cannot be the same as the response variable.

Component	Label	Function	Parameter 1	Parameter 2
1 (site effects)	1	Box-Cox power transform: $f(x) = \begin{cases} \ln x & \lambda = 0 \\ \frac{x^\lambda - 1}{\lambda} & \text{otherwise} \end{cases}$	λ	Not used
	2	Exponential transform: $f(x) = e^{ax}$	a	Not used
	3	Arctan transform: $f(x) = \arctan\left(\frac{x-a}{b}\right)$	a	b
	11–30	Fourier series representation of effect over the range (a, b). 11 and 12 are sine and cosine terms at the first Fourier frequency, 13 and 14 at second etc. *Odd* numbers correspond to *sine* terms (i.e. *odd* part of function). a and b can be specified *once only* for each site attribute. All sites must lie within the range $[a, b]$. Both a and b must always be treated as known.	a	b
	31–40	Legendre polynomial representation of effect over the range (a, b). 31 is linear, 32 is quadratic etc. a and b can be specified *once only* for each site attribute. All sites must lie within the range $[a, b]$. Both a and b must always be treated as known.	a	b
2 (year effects)	1	Linear: $f(t) = (t - 1950)/10$ (t is year)	No parameters required	
	2	Piecewise linear: $f(t) = \begin{cases} (t-a)/10 & \text{if } t > a \\ 0 & \text{otherwise} \end{cases}$	a	Not used
	3	Cyclical: $f(t) = -\cos\left(2\pi\frac{t-b}{a}\right)$	a	b

Table 3: Labels for nonlinear transformations of covariates (excluding previous days' values) in model definition files. This table should be used in conjunction with Table 1.

Label	Transformation
1	$\ln\left(Y_{t-k}^{(s)}\right)$
2	$\ln\left(1+Y_{t-k}^{(s)}\right)$
3	$I\left(Y_{t-k}^{(s)}>0\right)$ (i.e. indicator taking the value 1 if $Y_{t-k}^{(s)}$ was non-zero, 0 otherwise).
4	$I\left(0<Y_{t-k}^{(s)}<\tau\right)$, where τ is a 'trace' threshold (defined in row 8 of Table 1).
5	'Persistence' indicator: 1 if $Y_{t-1}^{(s)},\ldots,Y_{t-k}^{(s)}$ were all >0, 0 otherwise.
10–15	Transformations as above, but averaged over all sites with available data. Covariate is $S^{-1}\sum_r f\left(Y_{t-k}^{(r)}\right)$, where S is the number of contributing sites and $f(.)$ is the transformation. Code 10 is an average of untransformed values: $S^{-1}\sum_r Y_{t-k}^{(r)}$.
20–25	Transformations as above, but averaged over all sites with available data using weights that decrease exponentially with distance from the current site s. Covariate is $\sum_r w_{r,s} f\left(Y_{t-k}^{(r)}\right)$, where the weights $\{w_{r,s}\}$ sum to 1 and are proportional to $\exp\left[-ad_{r,s}\right]$. The value of a must be specified in the 'nonlinear parameters' section of the definition file — see Table 5.
30–35	Weighted averages of transformed values; weights proportional to $$\exp\left\{-a\left[(u_r-u_s-ku_0)^2+(v_r-v_s-kv_0)^2\right]^{1/2}\right\}.$$ See note 7, page 20 for an interpretation of this scheme.The values of a, u_0 and v_0 must be specified in the 'nonlinear parameters' section of the definition file — see Table 5.
110–115, 120–125 and 130–135	As 11–15, 21–25 and 31–35 but with the order of transformation and averaging reversed. Covariates are $f\left(\sum_r w_{r,s}Y_{t-k}^{(r)}\right)$ i.e. transformations of averages rather than averages of transformations.

Table 4: Labels for specifying nonlinear transformations of previous (and current, in a multivariate setting) days' values, in model definition files. $Y_t^{(s)}$ denotes the value at site s on day t; u_s and v_s are the geographical co-ordinates of site s (taken as the **first two** site attributes defined in the `siteinfo` object — see Section 3.7); and d_{s_1,s_2} is the distance between sites s_1 and s_2, again calculated from these two attributes. The expressions in the table relate to prediction of the value at site s on day t; k is the lag (in days), defined as described in Table 2.

2. In Table 2, the interpretation of Code 3 in Rglimclim differs from that in Glimclim. In Glimclim it was used to select a subset of cases for model fitting, based on the number of previous days' data available. In Rglimclim, this selection is performed using the argument `nprev.required` to `GLCfit()`. When reading a model definition file, by default Rglimclim will issue a warning if it finds something in this field, in case the user is accidentally using an old definition file. These warnings can be suppressed by passing the argument `oldGlimClim.warning=FALSE` to `read.modeldef()`.

3. In Table 2, covariates corresponding to a 'daily seasonal cycle' are calculated as though every year is a leap year. There is a tiny discontinuity between February 28th and March 1st in non-leap years.

4. In Table 2, the 'smooth month adjustments' are intended to allow departures from an overall seasonal cycle to be modelled smoothly, rather than via an indicator variable for a particular month (which results in a discontinuity in the fitted cycle). The adjustments are calculated from a shifted and scaled bisquare function

$$f(x) = \begin{cases} (1-x^2)^2 & |x| < 1 \\ 0 & \text{otherwise.} \end{cases}$$

   The scaling is chosen so that the adjustment takes its maximum value in the middle of the month, and is zero on the last day of the preceding month and on the first day of the following month.

5. In cases where the user wishes to define a covariate as a weighted average of previous values at all sites, any necessary parameters in the weighting schemes should be specified in the 'nonlinear parameters' section of the model definition file (see Tables 1, 4 and 5). Such parameters can be defined *only once* for each weighting scheme. Where more than one covariate is subject to the same weighting scheme (for example, if we wish to use weighted averages of values both one and two days ago), enter the index number of *any* of these covariates when defining the parameters. The software will automatically attach the correct weights to all other relevant covariates.

6. In row 8 of Table 1, there is an option to define a threshold below which positive-valued variables are simply regarded as 'small'. For non-negative variables such as rainfall and windspeed, the measurement of small values can be problematic. For data analysis purposes, perhaps the best way to deal with this problem is for the observer to set a threshold below which non-zero values cannot be measured accurately, and to record any non-zero values below this as 'trace' values. The analyst (and the software) can then treat these observations as censored data, and can adjust for the resulting uncertainty when fitting models.

   Unfortunately, observational practice tends to be inconsistent. At one site, the observer may be extremely diligent with regard to the reporting of trace values, while at another the observer may simply record them as zeroes. This can (and does) make fitted models appear to perform poorly on a site-by-site basis, because in its current form the software

is not able to model these 'human effects'. A pragmatic solution to this problem is to set any small values to zero, effectively reducing all of the data to the level of the least diligent observer. The question then arises: what should be done with the values *above* the threshold? The software allows 2 possibilities: 'soft' and 'hard' thresholding (the terminology is borrowed from the literature on wavelets). See Table 6 for details.

Note that the software will allow *only one* of these methods of dealing with small values. This restriction has been imposed deliberately, in case users are tempted to build nonsensical models.

When simulating models for thresholded data (codes 2 and 3 in Table 6), the software converts the simulations back to the scale of the original variable where necessary. Specifically:

- For 'soft' thresholding, after simulation the threshold is added back to any non-zero values. Therefore the simulated output will contain no values between zero and the threshold.
- For 'hard' thresholding, no correction is made. In particular, positive values below the threshold are *not* set to zero, so that the simulated output may contain values between zero and the threshold. This may be seen as a problem. However, from a modelling perspective it is certainly the correct thing to do because, in this case, non-zero values are modelled on the scale of the original data. If the fitted model is reasonable, the problem will be negligible. Conversely, if the problem is non-negligible then the fitted model is unreasonable. If this is the case, 'soft' thresholding should be used instead.

7. In Table 5, weighting scheme 3:

$$w_{r,s} \propto \exp\left\{-a\left[(u_r - u_s - ku_0)^2 + (v_r - v_s - kv_0)^2\right]^{1/2}\right\}$$

allocates the greatest weight to a location displaced from the current site by a vector (ku_0, kv_0). It may be appropriate when movement of weather systems (at an average velocity of $(-u_0, -v_0)$ units per day) can be identified from the available data.

8. In Table 7, option 1 is to model inter-site dependence via an empirical inter-site correlation matrix. If this option is chosen, the `GLCfit()` routine will write the estimated correlations to a file[7] that can be accessed subsequently by the `GLCsim()` routine to simulate dependent sequences. The nature of the correlations depends on the underlying model, as described in Section 3.2 above. Note, however, that these empirical correlations are not guaranteed to produce a positive definite correlation structure: in this case it may be necessary to fit one of the parametric spatial correlation models (options 3 through 20 in Table 7). See next note.

[7]The name of this file is set via the `cor.file` argument to `GLCfit`.

Weighting scheme	Parameters		
	1	**2**	**3**
1: Equal weights at all sites	No parameters required		
2: Distance-based exponential decay: $w_{r,s} \propto \exp\left[-ad_{r,s}\right]$	a	Not required	
3: Distance-based exponential decay with shift in origin: $w_{r,s} \propto \exp\left\{-a\left[(u_r - u_s - ku_0)^2 + (v_r - v_s - kv_0)^2\right]^{1/2}\right\}$	u_0	v_0	a

Table 5: Parameters in schemes for computing weighted averages of previous days' values. This table should be used in conjunction with Tables 2 and 4. $w_{r,s}$ is the weight associated with site r when predicting for site s. All other notation is the same as for Table 4.

Value of Code 2 (Table 1, row 8)	**Treatment of 'small' values**
1	Treat values as 'trace amounts'. This option is designed with rainfall in mind. Any 'small' value will be regarded as non-zero (and hence will count as a 'wet' day in a logistic regression model for rainfall, for example), but will be treated as a left-censored observation in any models for non-zero amounts.
2	'Soft' thresholding. If the original variable of interest is Y and the threshold is τ then models are fitted to Y^*, where $Y^* = 0$ if $Y < \tau$, $Y - \tau$ otherwise.
3	'Hard' thresholding. If the original variable of interest is Y and the threshold is τ then models are fitted to Y^*, where $Y^* = 0$ if $Y < \tau$, Y otherwise.

Table 6: Methods for dealing with 'small' positive values. The threshold below which a value is regarded as 'small' is defined as a 'global' quantity in the main model definition file (see row 8 of Table 1).

Table 7: Labels for specifying spatial structures in model definition files. See Section 3.2 for more details of the structures available. For the correlation-based structures, d_{ij} denotes the Euclidean distance between sites i and j in terms of the first two site attributes defined in the `siteinfo` object (see Section 3.7). This table should be used in conjunction with Table 1.

Label	**Model description**	**Parameter**			
		1	**2**	**3**	**4**
0 (default)	Independence	No parameters required			
1	Empirical correlations between each pair of sites	No parameters required (see note 8 on page 20)			
2	Constant correlation between each pair of sites: $\rho(i,j) = \rho$	ρ	Not used		
3	Exponential correlation function: $\rho(i,j) = \exp[-\phi d_{ij}]$	ϕ	Not used		
4	Correlation function $\rho(i,j) = \alpha + (1-\alpha)\exp[-\phi d_{ij}]$	ϕ	α	Not used	
5	Powered exponential correlation function: $\rho(i,j) = \exp\left[-\phi d_{ij}^{\kappa}\right]$	ϕ	κ	Not used	
6	Correlation function $\rho(i,j) = \alpha + (1-\alpha)\exp\left[-\phi d_{ij}^{\kappa}\right]$	ϕ	κ	α	—
21	Conditional independence given weather state X, which is 0 for a 'dry' day and 1 for a 'wet' day. $P(X=1) = 1 - P(X=0) = \alpha$, the mean of the site probabilities predicted by the occurrence model. When $X = x$, the log odds for a non-zero value at site i is $\ln\left(\frac{p_i}{1-p_i}\right) + x\ln a - \ln b(\alpha, p_i)$, where p_i is the probability of rain at site i according to a logistic regression model, and $b(\alpha, p_i)$ is chosen to make the unconditional probability at the site equal to p_i. This structure is valid only for logistic regression models.	$\ln a$	Not used		

Table 7: (continued).

Label	Model description	Parameter			
		1	2	3	4
22	Dependence induced by specifying a Beta-Binomial distribution for the number of wet sites on any day. The mean of the distribution, θ, is fixed at the mean of the individual site probabilities, and the shape parameter ϕ is estimated from data. This structure is only valid for logistic regression models.	ϕ	Parameters 2 and 3 are optional, and control program behaviour for days when the specified Beta-Binomial distribution is incompatible with the probabilities of rain at the sites. When this occurs, the probabilities are shrunk towards θ: p_i becomes $p_i - \lambda(p_i - \theta)$, for $\lambda \in (0, 1)$. The default value of λ is 0.01: to change this, enter it as parameter 2 for this model. By default, an error message will be printed to screen whenever such shrinking occurs. To suppress this, set parameter 3 equal to zero.		

9. In Table 7, many of the correlation-based structures (codes 3 through 20) are fitted by minimising a weighted least-squares criterion between the modelled and empirical correlations. The first step in estimating such structures is to calculate the empirical matrix and write to a file, as in the previous note. Parameter estimation is then carried out using an iterative numerical minimisation scheme (see Appendix E.2). The success of such fits can be dependent on a good choice of starting values in the model definition files. However, if the user enters a starting value of zero for any of the parameters in these correlation-based structures, then the software will set its own starting value using some potentially sensible heuristics.

4 Hints on use

This section contains a few guidelines which may be useful for avoiding error messages and meaningless models.

1. When fitting models, start with a very basic model (e.g. no covariates except a constant) and gradually increase its complexity, adding a couple of covariates at a time. The easiest way to do this is to use `write.modeldef()` to create a new definition file corresponding to the model that has just been fitted, and to edit this definition file. When adding new covariates whose coefficients are unknown, set the coefficient values to zero in the new model definition file.

   There are two reasons for this recommendation:

(a) The chances of error in defining a model are reduced — the user only has to make small changes to a model definition file at each stage.

(b) The approach should ensure computational stability, by providing reasonable starting values for fitting each model.

2. When building up a model, add 'obvious' covariates (for example those representing seasonality and temporal dependence) first. There is little doubt that such covariates should appear in a model (although it may be necessary to compare different representations of temporal dependence, for example), and it makes sense to start by getting close to a reasonable model quickly. Moreover, such a strategy makes it unlikely that any covariates added early on will subsequently need to be dropped.

3. For computational stability, covariate values should not be too large. For example, it might be necessary to express site altitude in hundreds of metres rather than in metres. As a rule of thumb, choose units of measurement so that covariate values tend to be between 0.1 and 10 in magnitude, if possible.

4. Keep track of the number of covariates involving regional effects (including interactions). If this number approaches the number of sites from which data are available, there is a risk that the model may be overfitted to these individual sites and may not be reproducable at other locations. One warning sign is the presence of extremely large and uninterpretable coefficients (usually, but not necessarily, relating to site effects) in a fitted model involving large numbers of orthogonal series components to represent site effects. This is likely to be the result of one or two sites which do not fit the general pattern, and may be an indication that data from these sites are suspect.

   As a general strategy for defining site effects, it may be useful to fit a model containing *no* site effects, and to plot a map showing the magnitudes of mean residuals at each site. This can be done using the `plot` method for fitted model objects (set the `which.plots` argument to `3`). If there is clear regional structure in this map that can be related to, say, site altitude, then clearly altitude should be included as a covariate. If there is other clear systematic structure, then the map can be used to guide the choice of orthogonal series representation. For example, if residuals are positive in the north of a study area and negative in the south, it will probably be approriate to include Legendre polynomials for site northings. If there is *no* systematic structure in the map, there is little point trying to include regional effects in a model! In this case, if many sites have large mean residuals there may be some data quality problems; it may be worth working with thresholded data (see row 8 of Table 1) to try and overcome these.

5. When estimating parameters in nonlinear transformations of covariates, it is worth proceeding in several stages. First, fix the unknown parameters at some 'plausible' level and estimate the optimal regression coefficients (βs) for that level. Next, free up a single parameter and estimate that together with the βs, holding the remaining nonlinear parameters fixed; carry on freeing up more parameters gradually. It may be necessary at some stage to fix some parameters which had previously been freed

because, despite the sophistication of the algorithm used, it can sometimes be extremely slow. In some cases it may be clear that, for all practical purposes, the algorithm has converged (keep an eye on the log-likelihood at each iteration, to determine this) but many iterations may be spent making rather small changes. If this occurs, it may be worth fixing all nonlinear parameters at their approximately optimal values, and running a final fit to obtain the correct βs for these values. If standard errors are required, they can be obtained by freeing up the parameters and running a further 'fit' with zero iterations.

Convergence difficulties may also indicate a silly model. Please try and resist the temptation to shoot the programmer until you are sure your model is reasonable! Avoid models that are too complex to be supported by the available data. For example, it is unlikely that weather system movement can be detected in daily data over small areas, in which case weighting scheme 3 in Table 5 should not be used.

6. Log-likelihoods (or deviances) can only be used to compare models that have been fitted to the same dataset. This is particularly relevant when comparing models that have different numbers of 'autoregressive' terms. Typically, missing observations will mean that a model involving, say, 2 previous days' values can be fitted to a larger subset of data than a model involving 3 previous days' values. The solution, in this case, is to fit the 2-day model using just those observations for which a 3-day model can be fitted (using the `nprev.required` argument to `GLCfit()`), and compare log-likelihoods based on this common subset of observations.

7. In an ideal world, likelihood ratio tests or deviance comparisons are preferable to t-tests (i.e. the comparison of an estimate with its standard error) for determining the statistical significance of terms in a model. The reason is that likelihood ratios automatically adjust for correlations among the covariates. However, any model comparison can only be made if the underlying calculations are correct. In particular, if models are fitted to a network of sites then inter-site dependence will usually invalidate the 'naïve' versions. The software reports both 'naïve' and 'robust' standard errors and likelihood ratio statistics; the robust versions are adjusted for inter-site dependence and are usually preferable, whereas the naïve versions allow comparison with results from other software packages. Note, however, that for the robust versions no attempt is made to adjust the log-likelihoods and deviances themselves. The theory underlying the adjustments is given in Appendix C.2 — note that the adjustments are independent of the particular spatial structure used in the model.

8. As far as 'autocorrelation' effects are concerned, averaging previous days' values over several sites is likely to lead to better performance than considering each site's history separately[8]. However, the computational load is dramatically increased by averaging over sites, particularly if parameters in weighting schemes have to be estimated. Also,

[8]From a physical point of view, day-to-day dependence is dictated by the movement of weather systems, which affect whole areas rather than single sites. From a mathematical perspective, averaging over previous days can go some way towards alleviating the problems caused by inter-site dependence.

there is a possibility of bias when averaging over previous days for which some sites have missing data (the software computes averages over all sites with available data, and only designates the resulting covariate as 'missing' if there is missing data at the single site of interest). This is particularly true of averages computed using the 'distance-based exponential decay with shift in origin' weighting scheme (Table 5) where edge effects may be present near the boundary of a study region. Such effects may manifest themselves via a change in the variance of residuals at boundary sites.

9. When simulating models that involve averages of previous days' values, choose the sites for simulation carefully. If simulation is carried out for a sparse subset of the sites originally used for model fitting, averages computed over this subset may have rather different properties from weighted averages computed over the entire network. In such cases, simulation results will be biased.

10. When choosing model structures, bear in mind that some options have been designed primarily for use in a specific situation. For example, the beta-binomial spatial dependence structure (option 22 in Table 7) is designed for the modelling of rainfall occurrence in regions that are small relative to the synoptic scale so that sites tend to be mostly wet or mostly dry: this scheme does not explicitly represent the decay of correlation with inter-site separation, because in small regions this decay is rather small and it is hydrologically more relevant to capture the high frequency of simultaneous occurrence. Such a scheme is not appropriate for use in larger catchments, however, where the distance dependence is more obvious: in this case, one of the correlation-based schemes (which are slow and inaccurate for small regions where the correlations are very high) should be used instead.

11. It is often required to simulate time series at locations for which data are not available. To assess the credibility of any simulated sequence, ideally one would compare its properties with those of an observed sequence. If data are not available however, this is not possible. One solution is to use multiple imputation to generate sequences at the new locations which are conditioned on all of the available data (inter-site dependence can be exploited here), and to compare the properties of the imputed sequences with those of the simulations. All of the inter-site dependence structures in Rglimclim have been designed in such a way that this multiple imputation can be carried out. It requires, however, that any `siteinfo` object passed to the `GLCsim()` routine must contain details of the sites for which data are available in addition to the sites for which simulate time series are required.

5 Example

In this section, we work through a simple example to illustrate the use of the software. It is an artificial example relating to daily rainfall over part of Ashdown Forest in Sussex, England. A map of the area can be found in Milne (1958). The data were actually generated

by simulating a GLM fitted elsewhere, with appropriate modifications. These simulated data exhibit many typical features of rainfall sequences in temperate climates. To make things more realistic, rainfall amounts less than 0.1mm have been set to 'trace' values and appear in the data as values of 0.05mm. Moreover, approximately 20% of the values are missing (at random).

It is recommended that you create a separate directory to work through this example, because several new files will be generated and you need to know where they are. If you are running R from a terminal (e.g. on a Unix system) then you should start R from within this directory. Otherwise (e.g. Windows users), start up R and then set the working directory to the desired location (in R for Windows, this can be done either using the `setwd()` command or using the `Change dir` option from the `File` menu).

Having started R and set the working directory if necessary, load Rglimclim:

```
> require(Rglimclim)
```

If you get an error here, it is probably because you haven't installed Rglimclim yet: see the instructions in Section 2. Otherwise, you're ready to start.

5.1 Prepare data

Referring back to Section 3.3, the first stage in any analysis is to collate the necessary data. In this case, the data are provided with the Rglimclim distribution: to access them, type

```
> data(GLCdemo)
```

at the R prompt. For a description of the data, type

```
> help(GLCdemo)
```

The data values themselves are stored in a data frame called `Ashdown.data`. To see its structure, type

```
> head(Ashdown.data,12)
```

```
   Year Month Day Site Rain
1  1970     1   1   G1 3.74
2  1970     1   1   G2 0.21
3  1970     1   1   G4 0.15
4  1970     1   1   G5 1.21
5  1970     1   1   G6 1.34
6  1970     1   2   G1 1.99
```

```
7  1970     1   2   G3 1.63
8  1970     1   2   G4 4.98
9  1970     1   2   G5 3.68
10 1970     1   2   G6 0.00
11 1970     1   3   G1 0.00
12 1970     1   3   G4 0.59
```

This should be fairly self-explanatory, with the possible exception of the `Site` column which contains site identifiers '`G1`','`G2`' etc. Note that the data are ordered by date and site, with missing observations excluded. For example, at the beginning of the file all 6 sites have data for 1st January 1970, but site 2 is missing for 2nd January.

As discussed in Section 3.4, Rglimclim needs to access data from files rather than from R objects. We must therefore write the data to a file with the correct format:

```
> write.GLCdata(Ashdown.data,file="Ashdown.dat")
```

By default, the `write.GLCdata()` routine assumes that the first four columns of `Ashdown.data` represent year, month, day and site code; and that any remaining columns are data values for different variables. This is the case here (the only remaining column is the `Rain` column; for multivariate modelling we could include additional columns as well). Finer control is available by specifying additional arguments to `write.GLCdata()`: consult the help page for more details.

You should now find that you have a file called `Ashdown.dat` in your working directory. Open it in a text editor to see its structure.[9] When finished, close the editor and move on to the next section.

5.2 Define site information

The data in this example are from six sites. Information on these sites is provided in a data frame called `Ashdown.sites`:

```
> Ashdown.sites
```

```
                          Name Region Eastings Northings
G1             Pooh Bear's House      1      1.0       4.0
G2       Where the Woozle Wasn't      1      2.5       1.5
G3     Sandy pit where Roo plays      1      4.0       6.0
G4 Rabbit's frends and raletions      2      6.5       5.0
G5        Eeyore's Gloomy Place      2      9.5       1.0
G6                      Bee Tree      2      7.5       6.0
```

[9] Windows users: the file is probably too large for Notepad to handle. A much better text editor for Windows is `Notepad++`, available from http://notepad-plus-plus.org/.

Again, this should be self-explanatory: site `G1` is called `Pooh Bears House` and is in Region 1 (see below), with geographical co-ordinates (1.0, 4.0). Note that the site identifiers `G1`, ..., `G6` here are not labelled as variables in the data frame — they are the row names. This is not necessary in general, however.

More information on the regions can be found in a data frame called `Ashdown.regions`:

```
> Ashdown.regions

  Region                                    Name
1      0                          Ashdown Forest
2      1   Pooh and Piglet's side of the forest
3      2 Christopher Robin's side of the forest
```

This simply contains one column of region numbers, and one number giving the corresponding names. The sole purpose of this is to provide interpretable labels for simulation output later on. We see that Region 1 (in which Pooh Bear's House can be found, according to the previous output) is called `Pooh and Piglet's side of the forest`. Region 0 corresponds to the entire area (i.e. all six sites).

The two data frames provided here cannot be used directly to pass site information to the Rglimclim fitting and simulation routines. To do this, we need to make an object of class `siteinfo`. This is easy to do:

```
> Ashdown.siteinfo <-
        make.siteinfo(Ashdown.sites,site.names=1,region.col=2,
        attr.names=c("Eastings (inches from left of 11\" wide map)",
                     "Northings (inches from bottom of 8\" high map)"),
        regions=Ashdown.regions)
```

The argument `site.names=1` here means that the names of the sites can be found in the first column of `Ashdown.sites`; `region.col=2` means that the second column contains the region codes; and the remaining columns are interpreted as attribute values for which full names are defined via the `attr.names` argument. By default, as in this example the `make.siteinfo()` routine takes the site identifiers from the row names of the data frame: see the help page for other options.

To check that the site information has been defined successfully:

```
> Ashdown.siteinfo

6 sites defined, each with the following attributes:

1. Eastings (inches from left of 11" wide map)
2. Northings (inches from bottom of 8" high map)
```

Detailed information is not written to screen here: users wishing to see the underlying object structure can do so by accessing its named components which are described in the help page for `make.siteinfo()`.

Note the following:

- For the site identifiers, justification is important — the code for site G1 here is 'space space G 1', and the software requires an exact match between the identifiers in the data file and in the site definitions. So for example ' `G1`' and ' `G1` ' would be interpreted as different sites[10].
- The first (and, in fact, only) two site attributes defined are the site X and Y co-ordinates. This follows the recommendations given in Section 3.8 above, and relates to the fact that the software will use these first two site attributes to compute inter-site distances if necessary.

5.3 Model fitting

5.3.1 Trivial rainfall occurrence model

Now that we have created a data file and defined our site information, we can start to fit models. Let's start with the simplest possible logistic regression model for rainfall occurrence. Let $p_{s,t}$ be the probability of non-zero rainfall for site s on day t. The model we will fit is

$$\ln\left(\frac{p_{st}}{1-p_{st}}\right) = \beta_0 \quad \text{or equivalently} \quad p_{st} = \frac{\exp[\beta_0]}{1+\exp[\beta_0]} \ . \tag{3}$$

This is a GLM in which the single covariate is a constant term. To fit it, we must first define the model structure to the system and then estimate the parameters (here just the coefficient β_0). To help you get started, the software comes with a model object already defined for this model structure, called `ConstantModel`:

```
> ConstantModel

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

[10] A useful trick, if at any stage you want to fit or simulate a model at a subset of sites, is to make small changes to the identifiers in a site definition object. For example, if you want to omit site `G3` from the analysis, change its code in the site definition to ' `?G3`'. The software will no longer recognise the data from this site in the data file, so it will be omitted from the analysis.

```
No dispersion parameters defined

Spatial dependence structure:
-----------------------------
Structure used: Independence
Warning message:
In print.GLC.modeldef(list(model.type = "logistic", Np = c(0L, 0L,  :
No global quantities (trace thresholds etc.) defined
```

Note the warning here: the model definition does not define a threshold below which positive values should be considered 'small'. Recall from the beginning of this section (page 26) that any value less than 0.1mm should be regarded as a trace, and appears in the data files as 0.05mm. Hence, in our modelling of these data, we should define this information to the system at the outset (in fact it makes no difference in this particular example, but it is good modelling practice).

To define the 'trace threshold' to the system, we will define a revised model structure using a definition file. The easiest way to do this is to start by writing the existing model structure to file:

```
> write.modeldef(ConstantModel,file="Model0_Init.def")

NULL
```

This creates a file called `Model0_Init.def`. Now, proceed as follows:

1. Open `Model0_Init.def` with a text editor outside R (Windows users: see previous comment about use of Notepad).
2. Scroll down to the model definition section at the bottom of the file. Note the row containing the model title `CONSTANT-ONLY MODEL`: this will be used to label software output as in the example above. The only other part of the model definition is the final line

   ```
   0    0.0000                Constant
   ```

   which defines the constant term in the model. This is line 49 of the file (if your editor does not display line numbers, throw it away and use a decent one!).
3. Tables 1 (row 8) and 6 tell us how to define a trace threshold to the system: add an extra line

   ```
   8    0.1000    1    1
   ```

to the end of the 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 quit the editor.

4. Read the new model definition into R:

```
> Model0.Init <- read.modeldef("Model0_Init.def",model.type="logistic",
                               siteinfo=Ashdown.siteinfo)
```

It may appear surprising that the model type and site information need to be specified at this stage. The reason is that for more complicated models these are sometimes needed to initialise the model structure.

If you get an error

```
Error in read.modeldef("Model0_Init.def", model.type = "logistic",
                       siteinfo = Ashdown.siteinfo) :
  Input error while reading line 51 of file Model0_Init.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 we 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:

```
> Model0.Init

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


Global quantities:
------------------
     Trace threshold:      0.1000

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. Some default values have also been substituted, relating to dispersion parameters and to inter-site dependence structure.

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

```
> Model0.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
+                     model.def=Model0.Init,data.file="Ashdown.dat",
+                     diagnostics=1,nprev.required=0)

Checking files ...
Reading data ...
Beginning estimation ...
      Iteration      Log-likelihood    Largest standardised score
      ---------      --------------    --------------------------
              0          -24393.236      27.0582  (parameter   1)
              1          -24025.899       0.1880  (parameter   1)
              2          -24025.881       0.0000  (parameter   1)
              3          -24025.881       0.0000  (parameter   1)
Computing covariance matrix of estimates ...
```

   The first four arguments to the `GLCfit()` command here are self-explanatory: we are fitting a logistic regression model to data from the sites defined in `Ashdown.siteinfo`, with a model structure defined in `Model0.Init` ad taking the data values themselves from file `Ashdown.dat`. Don't worry about the other two arguments for the moment — we will return to them later.

   The fitting is done by maximising a log-likelihood function obtained under the assumption that sites are independent of each other (see Appendix C for more details). The fitting routine writes progress to screen as it works: the fitting algorithm is iterative and, 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). In this case the algorithm converges quickly, yielding a maximised log-likelihood of -24025.881 at which the gradient is zero (confirming that this is indeed a turning point of the log-likelihood function).

7. Inspect the fitted model:

```
> Model0.fitted

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

Response variable: Y

Main effects:
-------------
                                  Coefficient  Std Err   Z-stat Pr(|Z|>z)
                        Constant       0.2905   0.0214  13.5749 < 2.2e-16
```

```
Global quantities:
------------------
     Trace threshold:      0.1000

No dispersion parameters defined

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

Notice the following: :

- The 'Coefficient' value of 0.2905 is the estimate of β_0 in **(3)**. The corresponding probability of rainfall on any day is $\exp(0.2905)/[1+\exp(0.2905)] = 0.572$. This is perhaps a long-winded way to discover that 57.2% of the values in the database are non-zero, but it does at least provide some insight into the model structure, and serves as a simple check on the software output!
- The value of 0.0214 in the 'Std Err' column is a robust standard error that accounts for the possibility of inter-site dependence. 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.

8. When fitting the model above, we passed the argument `siagnostics=1` to the `GLCfit()` command. This tells the routine to store some basic diagnostics to help check the model structure (alternatives are `diagnostics=0` to suppress the calculation of diagnostics, and `diagnostics=2` to produce an output file containing detailed diagnostic information for further analysis). One way to see these diagnostics is to type

```
> summary(Model0.fitted)

[Output suppressed]
```

The summary starts with some basic information about the fit, which is followed by some tables of diagnostic information. The first of these tables provides a way of checking the probability structure of the model. The basic idea is that if we collect together all days for which the forecast probability of rain is 0.1, then 10% of these days should have experienced rain. In practice we collect together all days for which forecast probabilities lie in the ranges $(0, 0.1), [0.1, 0.2), \ldots, [0.9, 1.0)$ and calculated the observed and expected numbers of wet days. A lack of agreement between in any cell of the table indicates a problem with the model. Clearly however, for such a simple model as this the information from such a table is not particularly useful (it indicates that there were 35192 days — i.e. all of them, because the modelled probability of rain

is the same every day — when the forecast probability of rain was between 0.5 and 0.6, and that the observed and expected proportions of wet days were both 0.572).

The remaining tables relate to Pearson residuals. If the model is correct, these all come from distributions with mean zero and the same standard deviation (usually 1). To illustrate how they may guide us in model-building, locate the following section of the output:

```
Pearson residual summaries by month
-----------------------------------
    # of days   Mean Std Dev S.E. mean
Jan      2986  0.158   0.964     0.036
Feb      2677  0.139   0.970     0.037
Mar      3023 -0.112   1.010     0.035
Apr      2908 -0.187   1.010     0.036
May      2951 -0.218   1.008     0.035
Jun      2904 -0.208   1.009     0.036
Jul      2980 -0.079   1.009     0.035
Aug      2990 -0.074   1.008     0.035
Sep      2897  0.030   0.995     0.036
Oct      2986  0.092   0.982     0.036
Nov      2871  0.212   0.945     0.036
Dec      3019  0.255   0.928     0.036
```

This gives the mean Pearson residual for each month of the year, together with standard errors (once again corrected for inter-site dependence — details of the correction are given in Appendix D). Notice that the means in months 3–8 are all negative (indicating overprediction by the model), whereas the remainder are positive. The clear systematic structure tells us that the model is inadequate. An obvious way to improve things is to add some seasonal structure to the model.

To check for unexplained structure in the model, perhaps an even easier approach is to use the Pearson residual tables to construct graphical diagnostics:

```
> if (dev.cur() == 1) x11(width=8,height=6)
> par(mfrow=c(2,2))
> plot(Model0.fitted,which.plots=1:2)
```

The first command here opens a new graphics window, unless there is already a graphics device open. The next command sets up a 2×2 array of plots, and the final `plot()` command produces the diagnostic plots.

The results are shown in Figure 1. The first plot is simply a graphical representation of the monthly mean Pearson residuals that we examined previously; the dashed lines show the range within which most of them would be expected to lie under the model. The unexplained

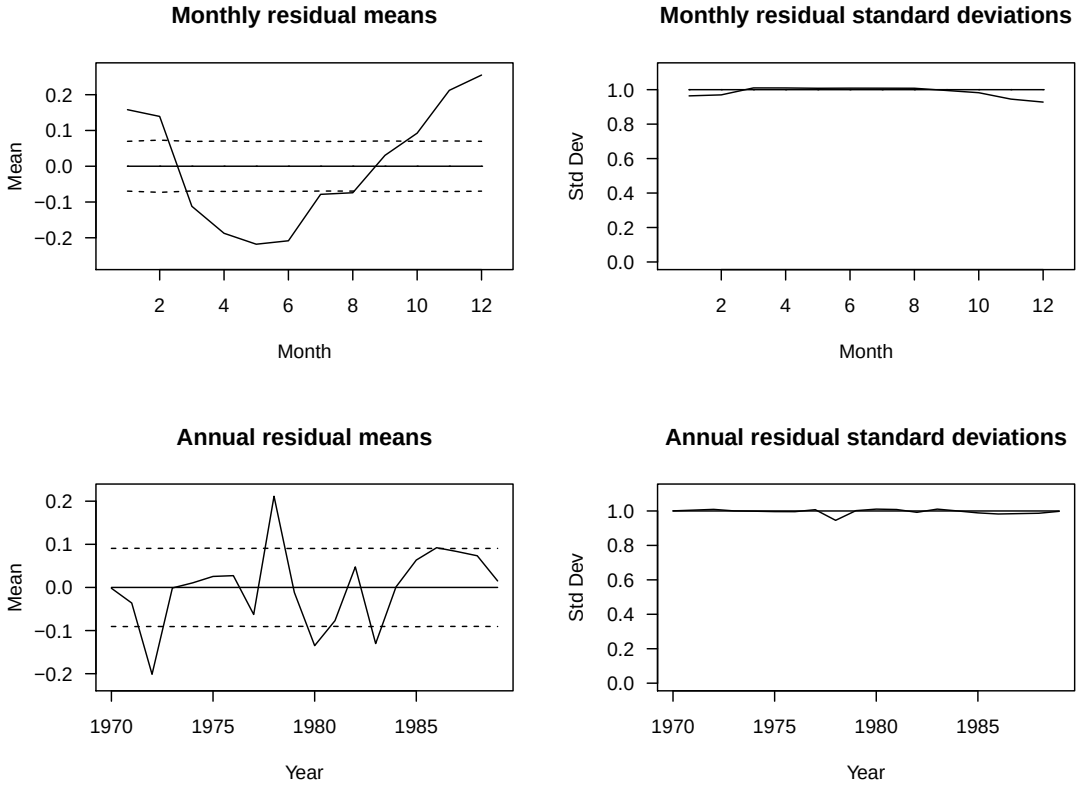

Figure 1: Diagnostic plots for initial logistic regression model.

seasonal structure is obvious. The second plot shows the monthly standard deviations, along with their expected value of 1. The plots and 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.

For more details on the options available for interrogating fitted model objects, type `help(GLC.modeldef)`.

5.3.2 Occurrence model with seasonality

We have now fitted a trivial logistic regression model, and established that it fails to capture the seasonality in the data. We therefore wish to extend this model by adding some seasonal structure. The steps are as follows:

1. Write the fitted model structure to a new definition file called `Model1_Init.def`:

```
> write.modeldef(Model0.fitted,file="Model1_Init.def")
```

```
NULL
```

2. Open this new definition file for editing. Notice that the value of the 'Constant' term is now 0.2905, as fitted previously.

3. Decide on a plausible representation of seasonality, from the options available in Tables 1 and 2. A good starting point is usually a Fourier representation of the annual cycle at a daily timescale. This requires both cosine and sine coefficients to be defined (since the phase of the cycle is unknown). Since the resulting covariates vary daily, we need to look at Table 2, and find that the required cosine term corresponds to a 'Code 1' value of 21. The sine term corresponds to a value of 22. So to define a simple annual cycle, insert the following two lines between the 'Constant' and 'Trace threshold' rows (recall from page 16 that rows must be ordered according to the value of `COMPONENT` 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, a value of zero is often a good starting point for estimation of coefficients in the fitting programs.

   Once again, take care with the alignment of the codes: there should be three spaced before each of the `21` and `22` so that these codes each occupy a total of five positions in their respective records.

   Finally, remember to update the title of the model — for example 'OCCURRENCE MODEL WITH SEASONALITY'. Save the modified definition file and quit the text editor.

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

```
> Model1.Init <- read.modeldef("Model1_Init.def",model.type="logistic",
+                              siteinfo=Ashdown.siteinfo)
> Model1.Init

OCCURRENCE MODEL WITH SEASONALITY
=================================
Main effects:
-------------
                                                          Coefficient
                                                 Constant      0.2905
1                       Daily seasonal effect, cosine component      0.0000
2                         Daily seasonal effect, sine component      0.0000

Global quantities:
------------------
     Trace threshold:      0.1000
```

```
No dispersion parameters defined

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

The software has interpreted the additional codes in `Model1_Init.def` as seasonal cosine and sine components (if you do *not* see the output above, you have made a mistake with your coding and should correct the definition file before proceeding).

5. Fit the updated model and inspect it:

```
> Model1.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
+                         model.def=Model1.Init,data.file="Ashdown.dat",
+                         diagnostics=1,nprev.required=0)

[output suppressed]

> Model1.fitted

OCCURRENCE MODEL WITH SEASONALITY
=================================

Response variable: Y

Main effects:
-------------
                                      Coefficient  Std Err   Z-stat Pr(|Z|>z)
                            Constant       0.2974   0.0216  13.7644 < 2.2e-16
1 Daily seasonal effect, cosine compon      0.3899   0.0305  12.7626 < 2.2e-16
2 Daily seasonal effect, sine componen     -0.2073   0.0305  -6.7884 1.134e-11

Global quantities:
------------------
     Trace threshold:      0.1000

No dispersion parameters defined

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

The p-values here indicate that all three coefficients differ from zero at any reasonable level of significance. Also, the maximised log-likelihood for this model is -23612.764 (this can be found in the `GLCfit()` output that has been suppressed above). The

log-likelihood for the previous model was -24025.881. The addition of two terms to the model has therefore increased the log-likelihood by 413.117. A formal test can be carried out to determine whether the data support the more complicated model (see Appendix B.1 for the underlying theory):

```
> anova(Model0.fitted,Model1.fitted)

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

Model 1: OCCURRENCE MODEL WITH SEASONALITY
Model 2: CONSTANT-ONLY MODEL

          Resid DF DF2-DF1      LogL     LLR          p Robust LLR   Robust p
M1           35189         -23612.76
M1 vs M2     35191       2 -24025.88 413.117 < 2.22e-16    105.103 < 2.22e-16
```

The interpretation of this output is as follows:

- The first row of the final table corresponds to the model with seasonality (note the use of the model titles to label the output), and the second row to the original model.
- The `Resid DF` column gives the residual degrees of freedom, usually defined as the number of observations less the number of parameters estimated. The `DF2-DF1` column shows the difference between the numbers of parameters estimated in the two models. The `LogL` column gives the maximised log-likelihoods, and the `LLR` ('log likelihood ratio') column is the difference between them. The column headed `p` is the p-value for a likelihood ratio test of the null hypothesis that the data were generated by the simpler model, under the assumption of no inter-site dependence. The remaining two columns give the log likelihood ratio and p-value after adjustment for this dependence. The tiny value of the `Robust p` here provides overwhelming evidence against the simpler model in favour of the more complex one.

This likelihood ratio test procedure is an alternative to the comparison of estimates with their standard errors. In general, it is to be preferred since it automatically adjusts for correlation among the covariates.

6. Examine the observed versus expected performance for the new model, but without displaying the tables of Pearson residuals:

```
> summary(Model1.fitted,tables=NULL)

OCCURRENCE MODEL WITH SEASONALITY
=================================
```

```
Response variable: Y

Model of type 'logistic', fitted to 35192 observations

# of parameters estimated: 3           Independence log-likelihood: -23612.76
Residual degrees of freedom: 35189     Deviance: 47225.53
Mean squared error (mean Brier score): 0.2391

No dispersion parameters estimated for this model

Pearson residuals: mean 1e-04 (std err 0.0106), standard deviation 1.0000

Occurrence frequencies vs forecasts:
------------------------------------
                                    Forecast decile
                   1     2     3     4     5     6     7     8     9    10
Observed prop.  0.000 0.000 0.000 0.000 0.482 0.547 0.647 0.000 0.000 0.000
Expected prop.  0.000 0.000 0.000 0.000 0.476 0.548 0.650 0.000 0.000 0.000
Number of cases     0     0     0     0  9221 11133 14838     0     0     0
```

There is now some variation in the wet-day probabilities, arising from the inclusion of seasonal covariates in the model. To see whether these covariates are able to explain all of the seasonality in the data, we can plot the results again:

```
> plot(Model1.fitted,which.plots=1:2)
```

The results are not shown here, but they suggest that there is still some seasonal structure remaining (negative means in March–May and August–October, and positive means elsewhere), albeit with a greatly reduced magnitude. The pattern may be due to some small misspecification of the cycle, or to some other covariate that has not been included in the model.

Of more concern here is the plot showing the mean Pearson residuals by year: although there is no obvious time trend, several years have mean residuals that fall substantially outside the limits expected under the model. Again, this indicates that we should search for additional covariates. Previous days' rainfalls are obvious candidates here, since rainfall sequences are generally autocorrelated in time. This autocorrelation will also affect the calculation of standard errors and likelihoods, so it is useful to account for it early on in a model-fitting exercise.

5.3.3 Rainfall occurrence — accounting for autocorrelation

The modelling of autocorrelation is achieved, within the GLM framework, by including previous days' values as covariates in a model. This poses a number of questions, for example:

how many previous days' values should be included? Can we benefit by transforming them? If so, what transformation should we use? Should we consider previous days' values at each site individually, or can we benefit by averaging over neighbouring sites as well?

In this tutorial, we will indicate how to go about answering the first two of these. Once users are familiar with the software, they will be able to answer the third as well. Proceed as follows:

1. Write the newly fitted model to a new definition file:

   ```
   > write.modeldef(Model1.fitted,file="Model2_Init.def")

   NULL
   ```

2. Open `Model2_Init.def` for editing. We'll start by adding a single previous day's rainfall, without any attempt at transformation. This covariate varies on a daily timescale, so `COMPONENT` has a value of 4. From Table 2, we need to put a '1' in the 'Code 1' field. So insert the following before the 'Trace threshold' row:

   ```
       4    0.0000    1
   ```

   Finally, give the new model a title before saving the definition file and quitting the text editor. For example, 'OCCURRENCE MODEL WITH SEASONALITY & Y[t-1]'.

3. Read the new definition file. To illustrate another feature of the software, we will add another argument to the `read.modeldef()` call this time:

   ```
   > Model2.Init <- read.modeldef("Model2_Init.def",model.type="logistic",
   +                        siteinfo=Ashdown.siteinfo,var.names="Rainfall")
   ```

   The additional argument here is `var.names="Rainfall"`. To see what this does, inspect the model definition:

   ```
   > Model2.Init

   OCCURRENCE MODEL WITH SEASONALITY & Y[t-1]
   ==========================================
   Main effects:
   -------------
                                                                  Coefficient
                                                     Constant         0.2974
   1                          Daily seasonal effect, cosine component  0.3899
   2                            Daily seasonal effect, sine component -0.2073
   3                                              Rainfall[t-1]        0.0000

   Global quantities:
   ```

```
------------------
     Trace threshold:      0.1000

No dispersion parameters defined

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

Notice that the third covariate is now labelled as `Rainfall[t-1]`: the software attempts to label the output as informatively as possible.

4. Run the fitting program and inspect the output:

```
> Model2.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
+                      model.def=Model2.Init,data.file="Ashdown.dat",
+                      diagnostics=1,nprev.required=0)

[output suppressed]

> Model2.fitted

[output suppressed]
```

The output shows that all four coefficients differ significantly from zero at any reasonable level of significance, and that the log-likelihood has increased dramatically from -23612.764 to -17954.621. However, if we try to carry out a formal test of this increase, we find a problem:

```
> anova(Model2.fitted,Model1.fitted)

Error in anova.GLC.modeldef(Model2.fitted, Model1.fitted) :
  Models were fitted to different numbers of observations
```

The problem here is that the latest model can only be fitted to observations for which the previous day's value is present in the dataset, whereas the previous model could be fitted to all observations. As noted at the start of this example, around 20% of the values are missing: thus there are considerably fewer observations available from which to fit the latest model. Formal statistical tests can only be used to compare models that have been fitted to the same data. This is the role of the `nprev.required` argument to `GLCfit()`: if we refit the previous model with `nprev.required=1`, it will be fitted using only those observations for which at least one previous day's value is available and the models can then be compared. We do not attempt this here since the coefficient associated with the previous day's rainfall is so overwhelmingly significant (and since there are good physical reasons for the presence of autocorrelation in rainfall time series).

At this point, we may want to know how many previous days' rainfall are relevant for predicting the current day's rainfall occurrence. We may also want to investigate whether previous days' rainfall amounts should be transformed prior to compare some transformations of this quantity: in particular, we may wonder whether knowledge of the amount of rain yesterday is more useful than just knowing *whether* it rained. We will defer any further residual analyses until we have finished modelling autocorrelation structure.

5. Edit `Model2_Init.def` again. At this stage, the penultimate line contains

```
4     0.0000     1
```

and defines the previous day's value. We now wish to define a transformation of this value. From Table 2, this can be achieved by inserting a value in the 'Code 2' field; and from Table 4, the value 3 defines a transformation that takes the value 1 for a non-zero amount and 0 otherwise. So change the penultimate line to:

```
4     0.0000     1     3
```

Save the file (overwriting the previous version), reread it, and refit the model as before to observations for which at least one previous day's value is available (note the change to the `nprev.required` argument):

```
>
> Model2a.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
+                      model.def=Model2.Init,data.file="Ashdown.dat",
+                      diagnostics=1,nprev.required=1)
```

The maximised log-likelihood is now −15527.415. This *is* directly comparable with the value of -17954.621 obtained above, since the models are fitted to the same dataset. Clearly, the new model is vastly superior to the old one, so the transformation is worthwhile. Of course, we could experiment with other transformations in Table 4 to find the one with the highest log-likelihood, but there is no further need of illustration here. We next try to establish how many previous days' values are needed in the model.

6. At this point we could simply expand the model to include an indicator for rainfall occurrence 2 days ago, then 3 days ago and so on. However, the effect of this is successively to add covariates that may be highly correlated. As a result, inference based on nominal standard errors can be misleading, and inference based on log-likelihoods is to be preferred. But since there is a lot of missing data, it is important to ensure that each model is fitted to the same dataset. For the sake of argument, let's restrict our search to models containing at most 4 previous days' values. The starting point is to refit our existing model to observations for which at least 4 previous days' values are available (note the `nprev.required=4` argument to the next command):

```
> Model2a.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
+                          model.def=Model2.Init,data.file="Ashdown.dat",
+                          diagnostics=1,nprev.required=4)

NULL
```

The final log-likelihood is -8001.956, although this is not directly comparable with any of the previous models because the present model is fitted to a reduced subset of the original data.

7. Write a new model definition file `Model2b_Init.def`:

```
> write.modeldef(Model2a.fitted,file="Model2b_Init.def")

NULL
```

Open the file for editing, add an extra line corresponding to an indicator for rainfall 2 days ago:

```
    4    0.0000    2    3
```

and change the model title ('OCCURRENCE MODEL WITH SEASONALITY, Y[t-1] & Y[t-2]', say). Then read the updated model definition:

```
> Model2b.Init <- read.modeldef("Model2b_Init.def",model.type="logistic",
+                               siteinfo=Ashdown.siteinfo,var.names="Rainfall")

Warning in read.modeldef("Model2b_Init.def", model.type = "logistic",
                         siteinfo = Ashdown.siteinfo,  :
  When defining contributions of previous days' values, meaning of code3
  is different in Rglimclim to that in previous versions of GlimClim
  (see manual for details). Set oldGlimClim.warning=FALSE to suppress
  this warning.
```

For more details on this warning message, see note 2 on page 19. It appears here because the previous `write.modeldef()` command has automatically inserted a value for Code 3 in the model definition file at line 52. It is unimportant here; however, following the advice of the warning message we will suppress it in the ensuing analyses.

8. Rerun the fitting routine:

```
> Model2b.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
+                          model.def=Model2b.Init,data.file="Ashdown.dat",
+                          diagnostics=1,nprev.required=4)

[output suppressed]
```

The maximised independence log-likelihood is now −7845.682. This is based on the same set of observations as the previous model incorporating just one previous day's rainfall. The two models can be compared formally, therefore:

```
> anova(Model2a.fitted,Model2b.fitted)

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

Model 1: OCCURRENCE MODEL WITH SEASONALITY, Y[t-1] & Y[t-2]
Model 2: OCCURRENCE MODEL WITH SEASONALITY & Y[t-1]

         Resid DF DF2-DF1      LogL     LLR          p Robust LLR   Robust p
M1          14570         -7845.682
M1 vs M2    14571       1 -8001.956 156.275 < 2.22e-16      80.95 < 2.22e-16
```

The robust p-value indicates an overwhelming rejection of the null hypothesis that the data were generated from the simpler of these two models.

9. Fit models including both three and four previous days' rainfall indicators, in a similar manner:

   (a) `> write.modeldef(Model2b.fitted,file="Model2c_Init.def")`

   and edit the file `Model2c_Init.def` as appropriate.

   (b)
   ```
   > Model2c.Init <- read.modeldef("Model2c_Init.def",model.type="logistic",
                          siteinfo=Ashdown.siteinfo,var.names="Rainfall",
                          oldGlimClim.warning=FALSE)
   ```

   [Output suppressed]

   ```
   > Model2c.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
                           model.def=Model2c.Init,data.file="Ashdown.dat",
                           diagnostics=1,nprev.required=4)
   ```

   [Output suppressed]

   (c) `> write.modeldef(Model2c.fitted,file="Model2d_Init.def")`

   and edit the file `Model2d_Init.def` as appropriate.

   (d)
   ```
   > Model2d.Init <- read.modeldef("Model2d_Init.def",model.type="logistic",
                          siteinfo=Ashdown.siteinfo,var.names="Rainfall",
                          oldGlimClim.warning=FALSE)
   ```

   [Output suppressed]

   ```
   > Model2d.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
                           model.def=Model2d.Init,data.file="Ashdown.dat",
                           diagnostics=1,nprev.required=4)
   ```

```
[Output suppressed]
```

You should obtain log-likelihoods of -7804.427 and -7803.805 respectively. This suggests that we should consider a model that incorporates just three previous days' indicators, since the fourth does not increase the log-likelihood significantly. To check this:

```
> anova(Model2a.fitted,Model2b.fitted,Model2c.fitted,Model2d.fitted)

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

Model 1: OCCURRENCE MODEL WITH SEASONALITY, Y[t-1] - Y[t-4]
Model 2: OCCURRENCE MODEL WITH SEASONALITY, Y[t-1] - Y[t-3]
Model 3: OCCURRENCE MODEL WITH SEASONALITY, Y[t-1] & Y[t-2]
Model 4: OCCURRENCE MODEL WITH SEASONALITY & Y[t-1]

         Resid DF DF2-DF1      LogL     LLR          p Robust LLR   Robust p
M1          14568         -7803.805
M1 vs M2    14569       1 -7804.427   0.622    0.26477      0.343    0.40745
M2 vs M3    14570       1 -7845.682  41.255 < 2.22e-16     22.595 1.7878e-11
M3 vs M4    14571       1 -8001.956 156.275 < 2.22e-16     79.673 < 2.22e-16
```

Note the following:

- In the table above, the models are ordered from the most complex to the most simple. At each stage, the null hypothesis being tested is that the data were generated from the simpler of the two models.
- The test for `M3` versus `M4` leads to an overwhelming rejection of `M4`: this suggests that more than one previous day's rainfall is required as a covariate in the model.
- The test for `M2` versus `M3` leads to a convincing rejection of `M3`, similarly: this suggests that more than two previous days' rainfalls are needed.
- With a robust p-value of around 0.4, the test for `M3` versus `M4` gives no reason to reject `M3`.

On the basis of these results, we may safely conclude that just three previous days are sufficient for our model. In this case, we may want to refit the 'three-day' model to include all cases for which three previous days' values are available so as to maximise the precision of our estimates (the fits above are limited to cases for which four previous days' values are available):

```
Model3.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
                    model.def=Model2c.Init,data.file="Ashdown.dat",
                    diagnostics=1,nprev.required=3)

[Output suppressed]
```

It may also be a good idea to revisit the model diagnostics:

```
> summary(Model3.fitted,tables=NULL)

OCCURRENCE MODEL WITH SEASONALITY, Y[t-1] - Y[t-3]
==================================================

Response variable: Rainfall

Model of type 'logistic', fitted to 18158 observations

# of parameters estimated: 6           Independence log-likelihood: -9734.6
Residual degrees of freedom: 18152     Deviance: 19469.21
Mean squared error (mean Brier score): 0.177

No dispersion parameters estimated for this model

Pearson residuals: mean -0.0011 (std err 0.0116), standard deviation 1.0002

Occurrence frequencies vs forecasts:
------------------------------------
                                    Forecast decile
                    1     2     3     4     5     6     7     8     9    10
Observed prop.  0.000 0.171 0.251 0.359 0.460 0.604 0.650 0.780 0.831 0.000
Expected prop.  0.000 0.190 0.245 0.354 0.457 0.587 0.661 0.768 0.837 0.000
Number of cases     0  1653  3328  1565  1121   394  1213  3297  5587     0
```

The tables of observed versus expected performance now show good agreement over a wide range of forecast probabilities. We should also check the representation of seasonal and annual structure:

```
> plot(Model3.fitted,which.plots=1:2)
```

These plots (not shown here) look much better than before: the seasonal structure in Pearson residuals has decreased, and almost all of the annual mean Pearson residuals now fall within the variability bands on the plots (one of the annual means falls outside the bands, but this is not a cause for concern given that 5% of the values are expected to lie outside the bands in any case). There is no discernible structure in the plots of monthly and annual standard deviations, either.

These results suggest that the model is starting to capture many features of the data. Obviously, we could experiment with the addition of other covariates representing temporal dependence (in particular, in rainfall sequences it is natural to consider the effects of 'persistence', which can be modelled via transformation 5 in Table 4). However, for the purposes of illustration we will now move on to consider interactions.

5.3.4 Rainfall occurrence — interactions

In northwestern Europe, winter rainfall tends to be produced by frontal weather systems that may last for several days. In summer however, there are more short-lived convective events. As a result, autocorrelation in rainfall sequences tends to be weaker in summer than in winter. Therefore, in a realistic model for rainfall occurrence, any parameters associated with previous days' rainfalls should themselves vary seasonally. Within a GLM, this can be achieved by defining interactions between previous days' rainfalls and seasonal covariates. We will use this example to demonstrate the software's capability for handling interactions.

1. Create a new definition file `Model4_Init.def`:

   ```
   > write.modeldef(Model3.fitted,file="Model4_Init.def")
   ```

   and open it for editing. All being well, the last 8 lines of the file should now read as follows:

   ```
   OCCURRENCE MODEL WITH SEASONALITY, Y[t-1] - Y[t-3]
       0   -1.2387                Constant
       4    0.2143   21           Daily seasonal effect, cosine component   1
       4   -0.1128   22           Daily seasonal effect, sine component     2
       4    1.8042    1    3     1I(Rainfall[t-1]>0)                       3
       4    0.5957    2    3     1I(Rainfall[t-2]>0)                       4
       4    0.3908    3    3     1I(Rainfall[t-3]>0)                       5
       8    0.1000    1    1      Trace threshold
   ```

   Note the following points:

   - When writing the updated definition files, the software has provided descriptive text in each row. This makes the definition files more readable.
   - At the right-hand end of each row is the covariate number. For example, the indicator for rainfall occurrence yesterday is covariate number 3. These numbers are required for defining both interactions and nonlinear transformations (you may have noticed that the covariate numbers also appear when you print the fitted models to screen).

   We wish to define interactions between the 'seasonal cycle' and 'previous days' covariates. This will result in the addition of six additional terms to the model — one for each seasonal/previous day combination. These are two-way interactions (each term involves two covariates). Referring to row 5 of Table 1, these are defined by entering the numbers of the interacting predictors in the 'Code 1' and 'Code 2' fields. So add the following lines before the trace threshold:

```
5    0.0000    1    3
5    0.0000    2    3
5    0.0000    1    4
5    0.0000    2    4
5    0.0000    1    5
5    0.0000    2    5
```

These define interactions between covariates 1 and 3, 2 and 3, ..., 2 and 5. Give your model a title, save and quit the editor.

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

```
> Model4.Init <- read.modeldef("Model4_Init.def",model.type="logistic",
+                     siteinfo=Ashdown.siteinfo,var.names="Rainfall",
+                     oldGlimClim.warning=FALSE)
> Model4.Init
```

The software splits the output into 'main effects' and '2-way interactions'. If you have defined the model correctly, the 'interactions' section should read as follows:

```
Two-way interactions:
---------------------
                                                    Coefficient
                        Daily seasonal effect, cosine component      0.0000
                                      with I(Rainfall[t-1]>0)
                          Daily seasonal effect, sine component      0.0000
                                      with I(Rainfall[t-1]>0)
                        Daily seasonal effect, cosine component      0.0000
                                      with I(Rainfall[t-2]>0)
                          Daily seasonal effect, sine component      0.0000
                                      with I(Rainfall[t-2]>0)
                        Daily seasonal effect, cosine component      0.0000
                                      with I(Rainfall[t-3]>0)
                          Daily seasonal effect, sine component      0.0000
                                      with I(Rainfall[t-3]>0)
```

Assuming your model definition is correct, run the fitting program to convergence and carry out a likelihood ratio test to see if the addition of interaction terms to the previous model is justified:

```
> Model4.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
+                     model.def=Model4.Init,data.file="Ashdown.dat",
+                     diagnostics=1,nprev.required=3)
> anova(Model3.fitted,Model4.fitted)
```

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

Model 1: OCCURRENCE MODEL WITH SEASONALITY, 3 PREVIOUS DAYS & INTERACTIONS
Model 2: OCCURRENCE MODEL WITH SEASONALITY, Y[t-1] - Y[t-3]

         Resid DF DF2-DF1      LogL   LLR        p Robust LLR Robust p
M1          18146         -9726.781
M1 vs M2    18152       6 -9734.604 7.823 0.015786      3.523 0.316636
```

The robust p-value of 0.317 does not provide any evidence for rejecting the null hypothesis that the data were generated from the simpler model without interactions. However, it is useful to look at the table of estimates and their standard errors (the main effects are omitted in the output below to save space):

```
> Model4.fitted

Two-way interactions:
---------------------
                                 Coefficient  Std Err   Z-stat Pr(|Z|>z)
 Daily seasonal effect, cosine co      0.0702   0.0809   0.8680   0.38541
          with I(Rainfall[t-1]>0)
 Daily seasonal effect, sine comp      0.1635   0.0811   2.0166   0.04374
          with I(Rainfall[t-1]>0)
 Daily seasonal effect, cosine co     -0.0744   0.0854  -0.8707   0.38392
          with I(Rainfall[t-2]>0)
 Daily seasonal effect, sine comp     -0.0879   0.0847  -1.0373   0.29959
          with I(Rainfall[t-2]>0)
 Daily seasonal effect, cosine co     -0.0350   0.0812  -0.4306   0.66672
          with I(Rainfall[t-3]>0)
 Daily seasonal effect, sine comp      0.0806   0.0808   0.9970   0.31879
          with I(Rainfall[t-3]>0)
```

The interactions involving Y_{t-2} and Y_{t-3} all appear insignificant at the 5% level. This suggests that we may try dropping these from the model. Of the two remaining terms involving Y_{t-1}, only one appears significantly different from zero. However, a seasonal cycle involves 2 parameters (phase and amplitude) and neither of these are known *a priori*, so from a modelling perspective it is good practice always to add seasonal components in pairs. We will keep both of these terms in the model.

3. Open `Model4_Init.def` for editing again; delete the four rows corresponding to the insignificant interactions, save and quit the editor. Read the definition file again, refit the model and retest:

```
> Model4.Init <- read.modeldef("Model4_Init.def",model.type="logistic",
+                    siteinfo=Ashdown.siteinfo,var.names="Rainfall",
```

```
+                     oldGlimClim.warning=FALSE)
> Model4.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
+                     model.def=Model4.Init,data.file="Ashdown.dat",
+                     diagnostics=1,nprev.required=3)
> anova(Model3.fitted,Model4.fitted)

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

Model 1: OCCURRENCE MODEL WITH SEASONALITY, 3 PREVIOUS DAYS & INTERACTIONS
Model 2: OCCURRENCE MODEL WITH SEASONALITY, Y[t-1] - Y[t-3]

         Resid DF DF2-DF1      LogL   LLR         p Robust LLR  Robust p
M1          18150         -9729.863
M1 vs M2    18152       2 -9734.604 4.741 0.0087263       2.07 0.1261446
```

Once again, the p-value suggests that the simpler model with no interactions is adequate. However, our understanding of European climate strongly suggests that these interactions *should* be present, and it does no harm to keep them in the model (given that the model is fitted to more than 18,000 observations, the inclusion of a couple of potentially redundant parameters is not going to cause problems) so we will retain them.

The diagnostics for this model, obtained using `summary()` and `plot()` (not shown here) do not give any cause for concern, with the possible exception of the block of negative monthly residuals between August and November (which could be accounted for by adding 'half-year cycles' to the 'seasonal' component of the model — Table 2, Code 1, values 23 and 24). Mean residuals are close to zero at all sites and for all years. There is possibly some disagreement between observed and expected rainday frequencies in the second and sixth deciles (probabilities in the ranges $[0.1, 0.2)$ and $[0.5, 0.6)$ respectively), but in practical terms this discrepancy is of little consequence.

5.3.5 Rainfall occurrence — site effects

So far we have not considered the possibility that there may be systematic regional variation in rainfall occurrence — it may tend to rain more frequently in the north of the study area than in the south, for example. One way to check for this is to produce an additional diagnostic plot for the current model:

```
> par(mfrow=c(1,1))
> plot(Model4.fitted,which.plots=3)
```

The result is shown in Figure 2, which shows the standardised mean Pearson residuals at each site.[11] The idea is that if there is any systematic regional variation that is not explained

[11] Notice that the axes are labelled using the site attribute descriptions that were defined earlier.

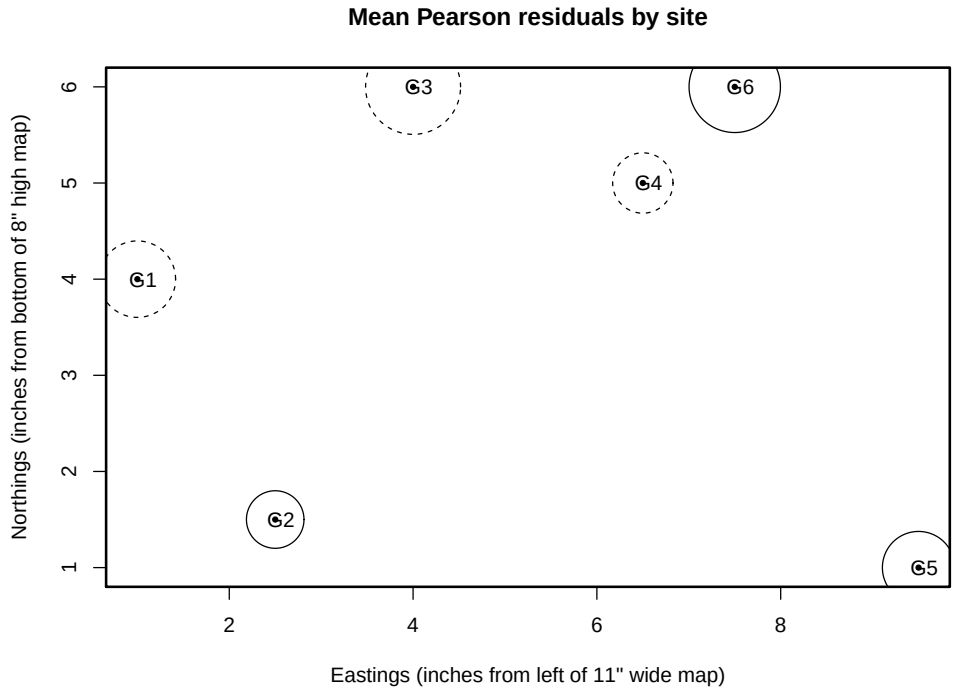

Figure 2: Bubble map showing mean residuals from fitted logistic regression model at each site. Circle areas are proportional to standardised mean residuals. Positive values are denoted by solid circles, negative values by dashed circles.

by the model then the sites with positive mean residuals (solid circles) will tend to cluster together, as will the sites with negative mean residuals (dashed circles). A lack of spatial organisation to the plot indicates that there is no systematic residual structure.

With only 6 sites in Figure 2, it is clearly not possible to identify complex patterns. In general, such a limited network may enable broad trends to be quantified, but little else. The map indicates a possible northwest-southeast gradient in the residuals (solid circles in the south and east, dashed circles in the north and west), that could be approximated by a planar surface. By adding such a structure to the model, we can determine whether this is a genuine effect, or merely a 'chance' configuration[12].

A planar surface may be represented as a linear combination of Eastings and Northings co-ordinates. For the present example, we could enter these directly into the model. However, we will instead take the opportunity to illustrate the use of Legendre polynomials of degree

[12]Strictly speaking, it is bad scientific practice to both derive and test a hypothesis using the same data. However, standard procedures can at least give some informal basis for judging the importance of a perceived effect. In particular, non-significant results (i.e. those indicating that an apparent pattern is merely due to chance) are probably reliable.

1 in each direction (the model formulation is equivalent, since a degree 1 polynomial is a linear transformation of the underlying quantity). Proceed as follows:

1. Create a new definition file `Model5_Init.def`:

   ```
   > write.modeldef(Model4.fitted,file="Model5_Init.def")
   ```

   and open it for editing. Check that the last 10 lines of the file are as follows:

   ```
   OCCURRENCE MODEL WITH SEASONALITY, 3 PREVIOUS DAYS & INTERACTIONS
       0   -1.2380                Constant
       4    0.1950   21           Daily seasonal effect, cosine component   1
       4   -0.1914   22           Daily seasonal effect, sine component     2
       4    1.8051    1    3     1I(Rainfall[t-1]>0)                        3
       4    0.5938    2    3     1I(Rainfall[t-2]>0)                        4
       4    0.3919    3    3     1I(Rainfall[t-3]>0)                        5
       5    0.0334    1    3      2-way interaction: covariates  1 and  3
       5    0.1500    2    3      2-way interaction: covariates  2 and  3
       8    0.1000    1    1      Trace threshold
   ```

2. Defining site effects now requires some care. Note the following:

   - Site effects (for which `COMPONENT` is 1 — see Table 1) must be defined *after* the constant term and *before* any other covariates. As a result, existing covariate numbers will change (for example, if we insert two rows corresponding to site effects then the existing covariate 1 will become covariate 3, and so on).
   - If there are interactions involving covariates whose numbers have changed, the corresponding interaction rows will need to be updated to reflect this change (forgetting to do this is a frequent source of errors!).
   - From row 1 of Table 1, site effect data are taken from the site information object `Ashdown.siteinfo` that we created earlier. Two attributes (Eastings and Northings) are defined in this object — either can be selected via the 'Code 1' field. Transformations (in our case Legendre polynomials) can be selected via an appropriate entry in the 'Code 2' field. Table 3 indicates that for a degree 1 polynomial we need a value of 31 in this field.
   - The Legendre polynomial representation requires, in addition to the degree of the polynomial, specification of the range (a, b) over which the representation holds. a and b are essentially parameters involved in the transformation of an underlying covariate, and hence can be defined via row 7 of Table 1.

   The desired model can therefore be specified by making the following changes to the definition file:

   (a) Insert two rows after the 'Constant' row:

```
1    0.0000    1   31
1    0.0000    2   31
```

Each of these defines a site effect: the first is transformation 31 of site attribute 1, the second is transformation 31 of site attribute 2.

(b) Edit the rows corresponding to interactions, so that they point to the correct main effects:

```
5    0.0334    3    5     2-way interaction: covariates  1 and  3
5    0.1500    4    5     2-way interaction: covariates  2 and  3
```

The software ignores the descriptive text when reading the file, so there is no need to update this unless you particularly want to.

(c) Add two rows after the interactions, to define the Eastings limits over which the Legendre polynomial representation is to hold. These should be chosen so that all sites fall within the limits. Based on the information in `Ashdown.sites`, we will set the limits to 0 and 11 respectively:

```
7    0.0000    1    1
7   11.0000    1    2
```

These rows define the first and second parameters, respectively, in the transformation of covariate 1 (see Table 3).

(d) Add two more rows to define the Northings limits (0 and 8):

```
7    0.0000    2    1
7    8.0000    2    2
```

(e) Change the title of the model.

The model definition section of the file should now look something like this:

```
OCCURRENCE MODEL WITH SEASONALITY, PREVIOUS, INTERACTION & SITE EFFECTS
    0   -1.2380              Constant
    1    0.0000    1   31
    1    0.0000    2   31
    4    0.1950   21         Daily seasonal effect, cosine component   1
    4   -0.1914   22         Daily seasonal effect, sine component     2
    4    1.8051    1    3    1I(Rainfall[t-1]>0)                       3
    4    0.5938    2    3    1I(Rainfall[t-2]>0)                       4
    4    0.3919    3    3    1I(Rainfall[t-3]>0)                       5
    5    0.0334    3    5     2-way interaction: covariates  1 and  3
    5    0.1500    4    5     2-way interaction: covariates  2 and  3
    7    0.0000    1    1
    7   11.0000    1    2
    7    0.0000    2    1
    7    8.0000    2    2
    8    0.1000    1    1     Trace threshold
```

Save the file, and quit the editor.

3. Read the model definition into R, and check it:

```
> Model5.Init <- read.modeldef("Model5_Init.def",model.type="logistic",
+                              siteinfo=Ashdown.siteinfo,var.names="Rainfall",
+                              oldGlimClim.warning=FALSE)
> Model5.Init
```

In particular, check the interactions and the section under 'Parameters in nonlinear transformations'. If there are any errors here (or if you get an error message), go back and correct the definition file. Otherwise, fit the model and compare with the previous one:

```
> Model5.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
+                         model.def=Model5.Init,data.file="Ashdown.dat",
+                         diagnostics=1,nprev.required=3)
> anova(Model5.fitted,Model4.fitted)

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

Model 1: OCCURRENCE MODEL WITH SEASONALITY, PREVIOUS, INTERACTION & SITE EFFECTS
Model 2: OCCURRENCE MODEL WITH SEASONALITY, 3 PREVIOUS DAYS & INTERACTIONS

         Resid DF DF2-DF1      LogL   LLR       p Robust LLR Robust p
M1          18148         -9728.521
M1 vs M2    18150       2 -9729.863 1.342 0.26131      1.862  0.15543
```

The robust p-value of 0.155 confirms our previous conjecture that no systematic regional patterns of rainfall occurrence are detectable in these data. Nonetheless, for present purposes it suffices to take this as our 'final' set of covariates. A residual analysis is, to all intents and purposes, indistinguishable from that of the previous model.

5.3.6 Rainfall occurrence: inter-site dependence

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 rainfall sequences from the fitted model in its current form, since the simulated sequences from each of the six sites will be independent. This is clearly unrealistic.

Rglimclim offers several alternative ways to model inter-site dependence in binary sequences. These are summarised in Table 7; more details are given in Appendix E.3. For

illustrative purposes we will consider the 'binary weather state' structure (label 21 in Table 7). The structure is explained in Section 3.2 and in Appendix E.3.

Proceed as follows:

1. Create a new definition file `Model6_Init.def`:

   ```
   > write.modeldef(Model5.fitted,file="Model6_Init.def")
   ```

   and open it for editing. Row 10 of Table 1 indicates how to define inter-site dependence structures. We wish to use structure 21, which involves a single parameter (Table 7). So append the following line at the end of the file:

   ```
   10     0.0000    21     1
   ```

   Give the model a title, and quit the editor.

2. Read the model definition into R and check:

   ```
   > Model6.Init <- read.modeldef("Model6_Init.def",model.type="logistic",
   +                      siteinfo=Ashdown.siteinfo,var.names="Rainfall",
   +                      oldGlimClim.warning=FALSE)
   > Model6.Init

   [output truncated]

   Spatial dependence structure:
   -----------------------------
   Structure used: Conditional independence given 'wet/dry' weather state
        Increase in logit on a 'wet' day:      0.0000
   ```

   If there are any errors at this stage, go back and correct them; otherwise, fit the model and store the result in `Model6.fitted`:

   ```
   > Model6.fitted <- GLCfit("logistic",siteinfo=Ashdown.siteinfo,
   +                         model.def=Model6.Init,data.file="Ashdown.dat",
   +                         diagnostics=1,nprev.required=3)

   [output suppressed]
   ```

   If you view the fitted model by typing `Model6.fitted`, you will find that the coefficients and standard errors are exactly the same as for the previous model; the only change is that the parameter of the spatial dependence model has changed (its estimated value is 6.5120).

This is now our final rainfall occurrence model, that can be used for simulation.

5.3.7 Modelling rainfall intensity

GLMs for daily rainfall usually consist of two parts: one uses logistic regression to model the occurrence of rain, and the other uses gamma distributions to model the rainfall intensity on 'wet' days. To model rainfall intensities here we will use a model structure that has been developed earlier, using a similar model-building strategy to that illustrated above for rainfall occurrence.

The structure is supplied as a model object called `Ashdown.IntensityModel.Initial`. Have a look at this, by typing `Ashdown.IntensityModel.Initial` at the R prompt (be careful: R is case-sensitive, so if you get an `object not found` error it is likely that you have mistyped something). Notice the following:

- One of the covariates is labelled `June indicator`. This is a variable taking the value 1 during June and zero elsewhere (code 16 in row 3 of Table 1), and has presumably been included because diagnostics indicated that without it the mean Pearson residual for June was significantly different from zero.
- Autocorrelation in this model is represented by including logarithmic transformations of previous days' rainfall amounts (transformation 2 in Table 4), rather than just indicators for previous days' rainfall occurrence. There is also a 'trace indicator', taking the value 1 if the previous day's rainfall was less than the trace threshold of 0.1mm and zero otherwise.
- The model has a dispersion parameter. For the gamma distribution, this is equal to the shape parameter (see Table 8 in Appendix A).
- Inter-site dependence is modelled via correlations between Anscombe residuals (see Appendix E.2). The correlations between each pair of sites are taken to be constant: this is presumably because the study area is so small that there is no discernible decay of the correlations with inter-site distance and hence there is little benefit from the use of a more complicated dependence structure.

We don't need to create a definition file, because the model object is ready to pass directly to `GLCfit()`:

```
> Intensity.fitted <- GLCfit("gamma",siteinfo=Ashdown.siteinfo,
+                            model.def=Ashdown.IntensityModel.Initial,
+                            data.file="Ashdown.dat",nprev.required=0,
+                            diagnostics=2,
+                            cor.file="IntensityCorrelations.dat",
+                            resid.file="IntensityResids.dat")

[output suppressed]
```

A few comments are in order here:

- By default, when fitting models of type `"gamma"`, `GLCfit()` will discard any zero values in the data files. The command above will therefore fit only to the non-zero rainfall amounts. This default behaviour can be overridden using the `response.check` argument to the `GLCfit()` command.
- The command above uses `nprev.required=0`, so that the routine will attempt to fit to all observations in the data file. However, because the model contains functions of three previous days' values as covariates, in fact the routine will only fit to observations for which all three values are available. This is handled automatically.
- We have now set `diagnostics=2` instead of `diagnostics=1`. The effect of this is to generate more extensive diagnostics for the fitted model, in the form of a file containing residual information for every case in the dataset (we will use this file below to generate a quantile-quantile plot of the residuals so as to check the gamma distributional assumption). The name of the file is specified in the `resid.file` argument.
- We have also used the `cor.file` argument: this specifies the name of a file to store the estimated correlations between each pair of sites, and is necessary because the model contains a correlation-based inter-site dependence structure.
- When fitting the model, you may notice some messages about reducing step sizes and a warning about the replacement of trace values by their approximate conditional expectations. These are not too important.

The fitted model can be displayed and interrogated in exactly the same way as the occurrence models that we have already studied. Some of the diagnostics are different however, because the response variable is continuous rather than binary and because we are using a correlation-based inter-site dependence structure. To see some of these additional diagnostics:

```
> par(mfrow=c(1,2))
> plot(Intensity.fitted,which.plots=4:5)
```

The results are shown in Figure 3. The left-hand plot is a quantile-quantile plot of standardised residuals under the fitted model (for the gamma distribution, the standardised residual for an observation Y_{st} is defined as Y_{st}/μ_{st} where μ_{st} is the modelled mean of the distribution; and, if the fitted model is correct, the standardised residuals should all come from the same gamma distribution with equal shape and scale parameter). The data values are shown as grey points, and the theoretical relationship under the model is shown as a dashed black line. The data are in excellent agreement with this theoretical relationship.

The second plot in Figure 3 shows the empirical inter-site correlations (grey dots) plotted against inter-site distance; the horizontal black line is the fitted correlation model. The

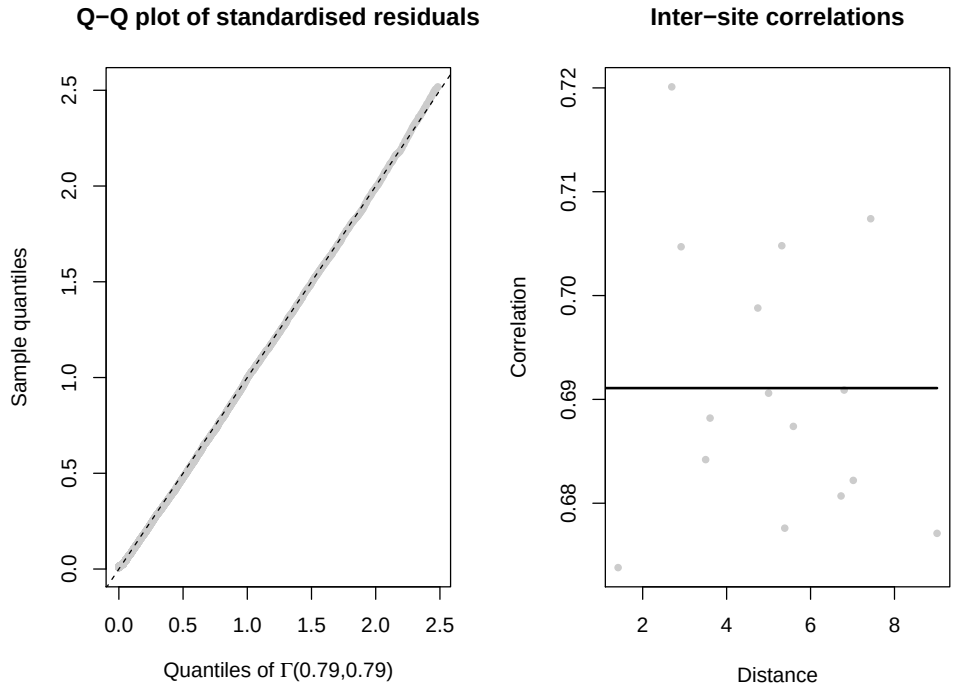

Figure 3: Some diagnostics for the fitted intensity model.

distances here are computed from the site attributed as defined in `Ashdown.siteinfo`. The plot enables us to assess whether the assumed correlation structure is adequate. Here, with so few sites it is difficult to tell: one could consider instead using a model in which the correlations decay exponentially with distance, but it is unlikely to affect the simulation performance dramatically (note, in particular, the data point in the bottom left-hand corner of this plot).

We have now fitted and checked our occurrence and intensity models, and can move on to perform some simulations. Before doing this however, take a quick look at the file `IntensityResids.dat` (generated by the `GLCfit()` command above). This can be opened in a text editor. The first few rows of the file are as follows:

```
SITE YEAR MONTH DAY OBSERVED PREDICTED        SD
  G1 1970     1   5   0.0618    3.3851    1.1219
  G5 1970     1   5   0.0617    3.2103    1.1219
  G1 1970     1   6   0.2100    2.2902    1.1219
  G5 1970     1   6   0.3000    2.1252    1.1219
  G1 1970     1   7   3.8500    3.3732    1.1219
  G2 1970     1   9   0.2800    3.1627    1.1219
  G6 1970     1   9   0.1900    2.7524    1.1219
```

Note the following:

- The first four columns are self-explanatory. The 'OBSERVED' column contains a observed rainfalls, 'PREDICTED' gives the means of the fitted gamma distributions; and 'SD' is the standard deviation of the standardised residual under the model.
- The file does not contain values for every site on every day. The excluded values are either those that could not be used in the fitting because they were missing or because one of the three previous days' values was missing; or those with rainfalls below the 0.1mm threshold that have been excluded from the intensity modelling.

This example should give the general feel of a model-building exercise using this software. We have not illustrated all of its features, but hopefully the user should now be reasonably familiar with the tables of codes in Section 3.8 above, and should be able to work out how to define more complicated models using the help pages. Although we have concentrated on logistic regression, the basic model-building process is the same for any GLM. The output varies slightly for different models, but the basic framework is always the same. For full explanations of the analysis methods, see the references given at the start of this manual.

5.4 Simulation

In this section we give a brief introduction to the simulation routine `GLCsim()`. We will simulate rainfall sequences using the occurrence and intensity models just fitted.

By working through the analyses above, you should now have objects `Model6.fitted` and `Intensity.fitted` in your R workspace. We will use these models to simulate 100 daily rainfall sequences at all six sites, for the 10-year period 1980 to 1999. This is easy:

```
> set.seed(2000)
> sim <- GLCsim(list(Occurrence=Model6.fitted,Intensity=Intensity.fitted),
+               nsims=100,start=198001,end=198912,impute.until=197912,
+               which.regions=0:2,simdir="./SimFiles",file.prefix="SimDemo")
```

The first command here sets the random number generator to a repeatable initial state, so that the results from the simulation can be reproduced exactly. The second carries out a simulation and stores information about this simulation in an object called `sim`. While it is running, note the following:

- The first argument to `GLCsim()` is a `list` containing both the occurrence and intensity models, named explicitly as `Occurrence` and `Intensity`. This naming is important: if the names were omitted, `GLCsim` would think that we were trying to perform a multivariate simulation in which one variable was modelled using `Model6.fitted` and the other using `Intensity.fitted`.

- The next three arguments `nsims`, `start` and `end` are self-explanatory. Note that the start and end dates are given in the form `YYYYMM` where `YYYY` is the year and `MM` is the month. The simulation will start on the first day of `start` and finish 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 time series of average rainfalls 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 subregions 1 and 2 as well.
- The routine will also generate a daily output file for each simulation; each of these output files has exactly the same format as the original data file (`Ashdown.dat`) that has been used throughout the modelling process. Together with the monthly output files therefore, this simulation will generate 200 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 what the routine does with this shortly.

It is worth noting that the `sim` object does *not* contain the simulated data: it merely contains information about what was simulated and where the data are stored. To see this:

```
> sim

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

Variables taken from data file Ashdown.dat
Variables simulated:
        1.        Rainfall (model type: logistic-gamma)

Simulation period: 1/1980 to 12/1989
No imputation performed
100 realisations generated

Output files generated: daily and monthly
Daily output written from 1/1980 to 12/1989
Monthly summaries written for the following regions:
        0       Ashdown Forest
        1       Pooh and Piglet's side of the forest
        2       Christopher Robin's side of the forest
```

```
Output directory: [your directory here]/SimFiles
Prefix for output filenames: SimDemo
```

Next, we can inspect the output files that have been generated:

```
> list.files(path="./SimFiles")

  [1] "SimDemo_Daily_Sim001.dat"   "SimDemo_Daily_Sim002.dat"
  [3] "SimDemo_Daily_Sim003.dat"   "SimDemo_Daily_Sim004.dat"
  [5] "SimDemo_Daily_Sim005.dat"   "SimDemo_Daily_Sim006.dat"

[output truncated]

[195] "SimDemo_Monthly_Sim095.dat" "SimDemo_Monthly_Sim096.dat"
[197] "SimDemo_Monthly_Sim097.dat" "SimDemo_Monthly_Sim098.dat"
[199] "SimDemo_Monthly_Sim099.dat" "SimDemo_Monthly_Sim100.dat"
```

Notice how the file names have been automatically generated, using the `file.prefix` argument that we supplied to the `GLCsim()` routine.

Next, open the first of the daily simulation files (`SimFiles/SimDemo_Daily_Sim001.dat`) in a text editor. If you set the random number seed to 2000 as above, the first few lines of this file will be as follows:

```
1980 1 1  G1  0.00
1980 1 1  G2 27.47
1980 1 1  G3  0.00
1980 1 1  G4  0.00
1980 1 1  G5 16.19
1980 1 1  G6  0.00
1980 1 2  G1  1.18
1980 1 2  G2  2.29
```

As noted above, the format of these daily simulation files is exactly the same as that of the original data file `Ashdown.dat`. For the simulated data however, of course there are no missing values.

Now open the monthly file `SimFiles/SimDemo_Monthly_Sim001.dat`. The first few lines should be[13]

```
1980  0  2.56 2.59 2.01 4.14 1.39 3.61 3.16 1.07 3.18 2.81 2.08 2.16 2.56
1980  1  2.80 1.60 2.39 3.97 0.85 3.72 3.02 1.11 2.73 2.81 1.96 2.01 2.41
1980  2  2.33 3.59 1.63 4.30 1.93 3.50 3.30 1.04 3.63 2.81 2.19 2.32 2.70
1981  0  3.68 2.03 2.61 2.79 1.45 2.29 1.18 2.26 2.34 1.82 3.32 3.34 2.43
1981  1  3.41 2.33 2.44 2.97 1.28 1.97 1.12 2.34 2.47 1.68 3.45 3.21 2.39
```

[13] The column separation has been reduced in this output, for reasons of space

The first two columns are the simulation number and region code (recall that we asked for summaries for all three regions, including region 0 which is the whole area). There follow 12 monthly means and an annual mean.

5.4.1 Multiple imputation

The daily (and possibly monthly) simulation files are now ready to be used for input into hydrological or other climate impacts models. However, to assess the credibility of the simulations it is helpful to be able to compare their with those of the observations. In doing so, one problem is that there are many missing values in the observations so that the observed properties cannot be known with certainty. One way to assess the resulting uncertainty is to generate additional simulated sequences in which the missing values are sampled from their conditional distributions given the available observations, and to calculate properties of interest for each of these additional simulated 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 39 imputations of the missing observations in our data set:

```
> obs <- GLCsim(list(Occurrence=Model6.fitted,Intensity=Intensity.fitted),
+               nsims=39,start=198001,end=198912,
+               which.regions=0:2,simdir="./SimFiles",file.prefix="Imputation")
```

The call to `GLCsim()` here is exactly the same as before, except for the following:

- 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` was set to a date prior to the start of the simulation period, in order to prevent the routine from doing imputation).
- The number of realisations is 39 rather than 100. The rationale for this is that if we calculate the value of any property of interest for each of these 39 simulations, the resulting range of values forms a 95% prediction interval for the actual value of that property that would have been observed given complete data (a proof of this assertion is left as an exercise for the interested reader ...).
- The `file.prefix` is now `Imputation`, so that we can distinguish the files containing simulations from those containing imputations.

We are now in a position to assess the simulation performance. This can be done by calculating a variety of summary statistics for each of the simulations, 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 rainfall for January. Each of

the 100 simulations will produce a different January mean rainfall, so that we have a simulated distribution of mean January rainfalls. The test of simulation performance is whether or not the *observed* mean rainfall can plausibly be considered to belong to this distribution.

Rglimclim offers a wide range of diagnostic plots for the assessment of simulation performance. To access these, it is first necessary to preprocess the simulation files and to calculate summary statistics for each simulation. We will calculate some summaries for the entire region: simulation performance at individual sites can also be explored if required (type `help(summary.GLCsim)` for full details). The preprocessing can be carried out as follows:

```
> seasons <- list(3:5,6:8,9:11,c(12,1,2))
> sim.summary <-
+         summary(sim,season.defs=seasons,thresholds=0,which.regions=0)

[output suppressed]

> obs.summary <-
+         summary(obs,season.defs=seasons,thresholds=0,which.regions=0)

[output suppressed]
```

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 indicates what 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 in the current setting, where we are studying rainfall and the threshold is set to zero). Notice also the use of the `season.defs` argument: this will compute annual time series of seasonal means for user-define 'seasons' (which are just groups of months, specified here in the `seasons` object).

Clearly, there is a lot of information in these summaries and it is hard to process all of it. One might reasonably examine some plots for selected sites, regions and statistics. For example:

```
> par(mfrow=c(2,5))
> plot(sim.summary,imputation=obs.summary,which.sites=NULL,
+                                         which.timescales="daily")
```

The results are shown in Figure 4. There are 10 plots here, each showing the simulated distributions of a different summary of the daily rainfall time series for each month of the year. The grey bands indicate the percentiles of the simulated distributions (see the figure caption for details) while the black bands are 95% intervals for the observed values (obtained

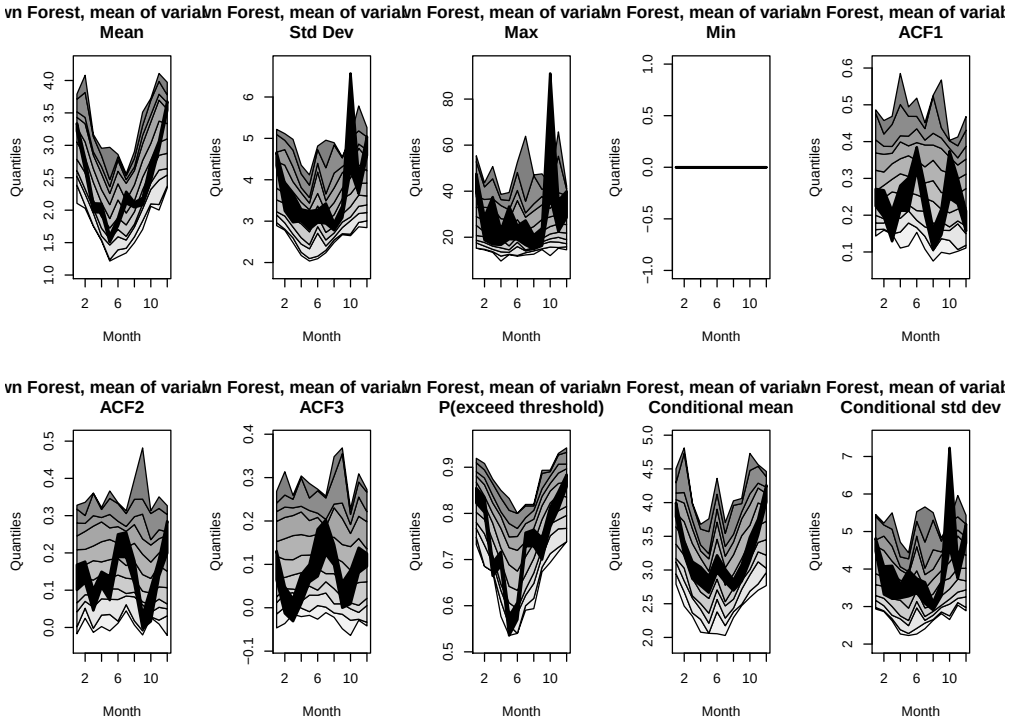

Figure 4: Simulated distributions of selected summary statistics for the entire study area, together with envelopes obtained from 39 imputations. The bands in the simulated distributions indicate the minimum and maximum values, together with the 1st, 5th, 10th, 25th, 50th, 75th, 90th, 95th and 99th percentiles.

from the range of values in the 39 imputations, as discussed above). The plots are too small to accommodate their titles here; for finer control over the plot titles, and the selection of which plots are produced, type `help(plot.summary.GLCsim)`. 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 every property 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). This seems to be the case throughout Figure 4, with the possible exception of the standard deviation, conditional standard deviation and maximum in October. However, for all of these statistics the imputation range indicates substantial uncertainty due to missing data: overall there, the simulations seem to do a good job of reproducing all of these properties.

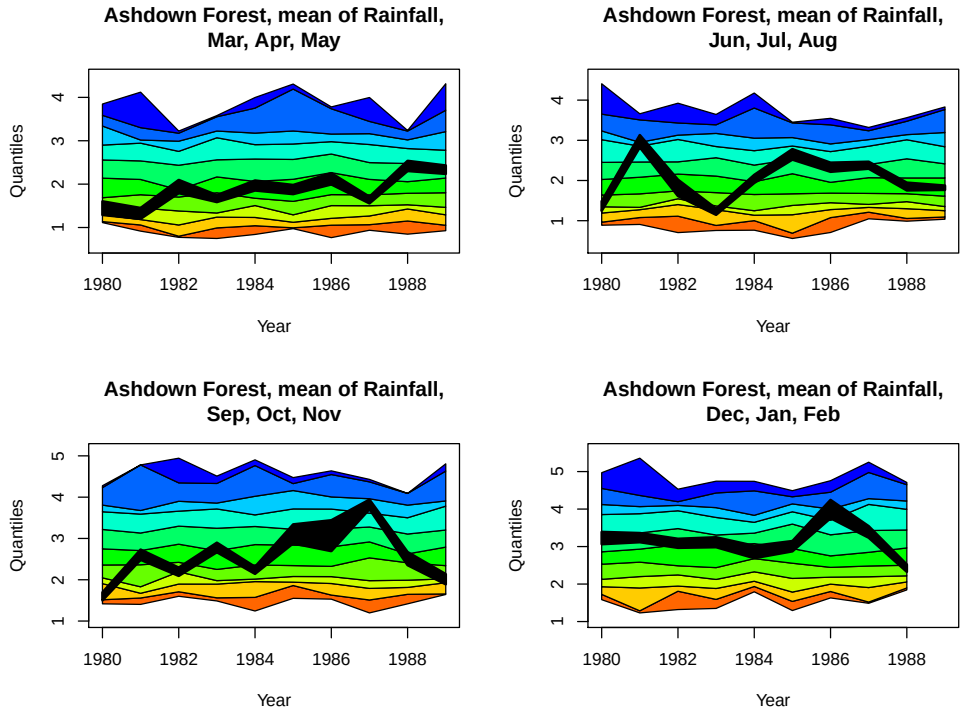

Figure 5: Simulated distributions of annual time series of seasonal mean rainfalls for the entire study area, together with envelopes obtained from 39 imputations.

In some applications, it is important to reproduce the variation in rainfall 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 this time setting `which.timescales="monthly"` rather than `"daily"`:

```
> par(mfrow=c(2,2))
> plot(sim.summary,imputation=obs.summary,which.sites=NULL,
+                                          which.timescales="monthly",
+                                          colours.sim="colour")
```

The result is shown in Figure 5. Once again, the simulations seem to do a good job of reproducing the observed variability. Note that the final plot stops at 1988 rather than 1989: this is because the simulation period ended in December 1989 so that simulated data for the final (December, January, February) season are not available.

This completes the tour of the software. Any bug reports, suggestions for improvements and general comments will be most welcome — email me at `richard@stats.ucl.ac.uk`. Meanwhile: good luck, and happy modelling!

Richard Chandler

Acknowledgements

The development, testing and documentation of this software has been partially supported by: the Office of Public Works, Dublin; the TSUNAMI consortium (grant A2P03); the Ministry of Agriculture, Fisheries and Food, London; the Department for the Environment, Food and Rural Affairs (R & D projects FD2103 and FD2105); and the Natural Environment Research Council (grant number NE/1006656/1). I am grateful to several colleagues for extremely helpful discussions over the years, in particular Valerie Isham, Howard Wheater, Vern Farewell, Steven Bate and Zhongwei Yan. Chiara Ambrosino has suffered greatly during the process of porting the package from its original Fortran version into R, and I am very appreciative of her patience as well as the code that she has contributed. Thanks must also go to the numerous people who have grappled with the software in the past and have found errors or made constructive suggestions for improvement: Claudia Annoni, Liz Baitson, Nadja Leith, Jing Liu, Amit McCann, Nick Price, Miew Ling Soo, Neeraj Teeluck, Chi Yang, Yoko Yoneyama and possibly others who are omitted due to inept-, rather than ingrat-, -itude (on my part, that is).

Technical Appendix

This appendix gives, for the interested user, technical details of the models and algorithms used in this software. It is necessarily brief and collates material that may be found in Cox and Hinkley (1974); Jørgensen (1983); Liang and Zeger (1986); McCullagh and Nelder (1989); Press et al. (1992); Fahrmeir and Tutz (1994); Wei (1997) and Dobson (2001), among others. For an overview and more comprehensive reference list, see Chandler and Wheater (2002), Yan et al. (2002) and Wheater et al. (2000, Chapter 4).

A Generalised Linear Models and the exponential family of distributions

A GLM, for a $n\times 1$ vector of random variables $\mathbf{Y} = (Y_1, \ldots, Y_n)'$, is a model for the probability distribution generating $\mathbf{Y}$. Each of the Ys is considered to depend on p covariates, whose values can be assembled into a $n \times p$ matrix $\mathbf{X}$ (the (i, j)th element of $\mathbf{X}$ is the value of the jth covariate for Y_i). The distribution of $\mathbf{Y}$ has vector mean $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)'$, which is related to $\mathbf{X}$ via the relationship

$$g(\boldsymbol{\mu}) = \mathbf{X}\boldsymbol{\beta} = \boldsymbol{\eta} \ , \text{ say.} \tag{4}$$

Here, $g(.)$ is a monotonic function (the LINK FUNCTION) and $\boldsymbol{\beta}$ is a $p\times 1$ vector of coefficients (by $g(\boldsymbol{\mu})$ we mean the $n \times 1$ vector whose ith element is given by $g(\mu_i)$). The elements of $\boldsymbol{\eta}$ are called LINEAR PREDICTORS.

For computational and inferential reasons, the distribution of each Y_i is restricted to belong to the EXPONENTIAL FAMILY. For current purposes, this may be defined as the family of all distributions with densities of the form

$$f(y; \psi, \phi) = \exp\left[\frac{y\psi - b(\psi)}{a(\phi)} + c(y, \phi)\right] \ , \tag{5}$$

for some parameters ψ and ϕ, and functions $a(.)$, $b(.)$ and $c(., .)$. Many standard distributions are in this family; some examples are given in Table 8.

For a distribution expressed in this way, the mean is $\partial b/\partial\psi$ and the variance is $a(\phi)\partial^2 b/\partial\psi^2$. These expressions may be verified readily for the examples given in Table 8. For the gamma distribution, for example, we have $b(\psi) = \ln\psi$, so that $\partial b/\partial\psi = \psi^{-1} = \mu$. For the variance, we get $a(\phi)\partial^2 b/\partial\psi^2 = (-\phi^{-1})\,(-\psi^{-2}) = \mu^2/\nu$.

These results suggest that the parameter ψ determines the mean of the distribution and that, given ψ, the parameter ϕ determines the variance. For this reason, ϕ is known as a DISPERSION PARAMETER. Equation (4) indicates that, in a GLM, the primary interest is in the relationship between the mean and the covariates. Hence, from the perspective of exponential families, ψ is a function of the covariates and of the coefficient vector $\boldsymbol{\beta}$. The

Distribution	**Density**	$\boldsymbol{\psi}$	$\boldsymbol{\phi}$	$\mathbf{a}(\boldsymbol{\phi})$	$\mathbf{b}(\boldsymbol{\psi})$	$\mathbf{c}(\mathbf{y},\boldsymbol{\phi})$
Bernoulli, parameter p	$p^y(1-p)^{1-y}$ $(y=0,1)$	$\ln\left(\frac{p}{1-p}\right)$	1	1	$\ln(1+e^{\psi})$	0
Poisson, parameter μ	$\left(e^{-\mu}\mu^y\right)/y!$ $(y\in\mathbb{N})$	$\ln\mu$	1	1	e^{ψ}	$-\ln y!$
Normal, parameters μ and σ^2	$\frac{1}{\sigma\sqrt{2\pi}}e^{\left[-(y-\mu)^2/2\sigma^2\right]}$ $(y\in\mathbb{R})$	μ	σ^2	ϕ	$\frac{\psi^2}{2}$	$-\left(\frac{y^2}{2\phi}+\frac{1}{2}\ln 2\pi\phi\right)$
Gamma, mean μ and shape parameter ν	$\left(\frac{y\nu}{\mu}\right)^{\nu}e^{-\nu y/\mu}/y\Gamma(\nu)$ $(y>0)$	μ^{-1}	ν	$-\phi^{-1}$	$\ln(\psi)$	$\phi\ln\phi y$ $-\ln y$ $-\ln\Gamma(\phi)$

Table 8: Some distributions in the exponential family.

dispersion parameter ϕ is usually assumed constant in a GLM (for example, for a normal distribution the dispersion parameter is the variance σ^2, which is assumed constant in a classical linear regression).

A.1 Interactions

It is not uncommon for covariates to interact with each other, by which we mean that the effect of one covariate may depend on the values of others. In North-Western Europe, for example, the impact of the North Atlantic Oscillation upon rainfall is confined mainly to the winter months. Hence the coefficient associated with the NAO in a GLM should vary seasonally. This can be achieved by representing the coefficient itself as a linear combination of covariates representing seasonality. Mathematically, this is equivalent to adding an extra covariate to the model, whose value is the product of the interacting covariates. Hence interactions can be incorporated straightforwardly within the overall framework of model (4).

A.2 Multivariate modelling

Suppose now that instead of an $n \times 1$ data vector we have an $n \times m$ data *matrix* $\mathbf{Y}$, each column of which represents the observations of a different variable so that the (i, j)th element Y_{ij} represents the value of the jth variable for the ith case in the dataset. If the distributions of the individual variables (possibly conditioned on covariates) are all Gaussian, then potentially we could build a regression model for all of the variables simultaneously. However, in practice this would be a gargantuan task for all but the simplest situations (the example in Section 5 shows the complexity of building a model for a single variable; to carry out this kind of analysis for several variables simultaneously would be prohibitive). Moreover, there are few tractable multivariate models that can be used in settings where one or more of the variables has a non-Gaussian distribution.

An alternative strategy for multivariate modelling is based on a standard factorisation of the joint distribution. Denote by $\mathbf{Y}_i = (Y_{i1}\ Y_{i2}\ \ldots\ Y_{im})'$ the collection of all variables for the ith case in the dataset, and let $\mathbf{x}_i$ be the associated vector of covariates. Then the joint density of $\mathbf{Y}_i$ can be factorised as

$$f(\mathbf{y}_i|\mathbf{x}_i) = f_1(y_{i1}|\mathbf{x}_i) \times f_2(y_{i2}|y_1, \mathbf{x}_i) \times \ldots \times f_m(y_{im}|y_1, y_2, \ldots, y_{m-1}, \mathbf{x}_i) \ ,$$

where here $f_\cdot(a|b)$ denotes the density of a conditional upon the value(s) of b. This factorisation enables a multivariate model to be constructed one variable at a time: start by developing a GLM for the first variable, conditioned upon the covariates, then develop a GLM for the second variable conditioned on the covariates *along with the first variable*, and proceed in this way until all variables have been modelled. Simulation of the fitted models can then also be done one variable at a time, generate a value of the first variable, then sample the second from its conditional distribution given the value of the first, and so on.

Note that in this framework, the model for each variable must not include as covariates any other variables that have not yet been modelled: this would create a "circular dependency" (such that A depends on B and B in turn depends on A, for example) which prevents the use of the simulation procedure just described. The `GLCsim()` routine checks for such circular dependencies and terminates if any are found.

An obvious question here is: does the order of the variables matter? The answer is yes. To see this, imagine building a bivariate model for rainfall and temperature. A normal-heteroscedastic model will often be reasonable for daily temperatures, while a logistic regression model is often used to describe rainfall occurrence as in Section 5. If rainfall is taken as the first variable, and temperature is conditioned on rainfall occurrence, then the implied marginal temperature distribution for each day will be a mixture of two normal distributions (one for dry days, and the other for wet): this mixture could be bimodal. On the other hand, if temperature is taken as the first variable then its modelled marginal distribution will be normal and unimodal.

Given that the order of the variables matters, it is worth giving some guidance as to how one might decide on an appropriate ordering. There is no unique answer to this. However, the following considerations may be helpful:

- If data availability permits, order the variables so as to respect any physical mechanisms that are present. For example, when building a model for a collection of variables including pressure, wind speed, rainfall, cloud cover and incoming radiation, it makes physical sense to start with pressure since the pressure field imposes a fundamental control on atmospheric dynamics. Wind speed is naturally modelled as dependent upon the pressure field, as are rainfall and cloud cover. Rainfall might be considered as additionally dependent on cloud cover, given that rain is unlikely to occur without clouds; and incoming radiation is also dependent on cloud cover.

  It is perhaps worth noting that many existing multivariate weather generators start with precipitation as the 'primary' variable and then derive other variables from this. This is a habit that dates back to the early development of weather generators in the 1980s, when there was little awareness of techniques such as GLMs outside the statistical literature and when precipitation generators were largely restricted to models based on Markov chains for which it was difficult to incorporate covariate information. The present GLM framework overcomes this limitation (indeed, Markov chain models can be considered as a special case) and hence provides the opportunity to develop multivariate models based on physical understanding rather than mathematical constraints.

- If data are plentiful for some variables but sparse for others, as a pragmatic strategy it makes sense to model the data-rich variables first. This is because a GLM can only be fitted to the cases for which observations on both the response and covariates are available. If the data-rich variables are modelled first then their models can be fitted to all of the available data for these variables. If they are modelled as conditional upon the data-poor variables however, far fewer observations will be available.

These considerations are often directly conflicting: for example, station-based pressure observations can be very sparse which may preclude the use of a physically-based ordering of the variables. Scientific judgement is required, therefore.[14]

B Maximum likelihood estimation

Given a data vector $\mathbf{y} = (y_1, \ldots, y_n)'$, assumed to be drawn from some family of distributions indexed by a parameter vector $\boldsymbol{\theta}$, we may wish to estimate this parameter vector. A standard way to do this is via maximum likelihood. The basic idea is to choose the value of $\boldsymbol{\theta}$ which allocates highest probability to the observations $\mathbf{y}$. Specifically, denote the joint density of $\mathbf{y}$ by $f(\mathbf{y}; \boldsymbol{\theta})$. Then the LIKELIHOOD for $\boldsymbol{\theta}$ given $\mathbf{y}$ is defined as

$$L(\boldsymbol{\theta}|\mathbf{y}) = f(\mathbf{y}; \boldsymbol{\theta}) \ , \tag{6}$$

[14] The use of gridded data products to 'get around' the problem of sparse data is most definitely *not* required: many of these data products do not get around the problem at all, they merely hide it.

and the MAXIMUM LIKELIHOOD ESTIMATE (MLE) of $\boldsymbol{\theta}$ is the value maximising this expression. Equivalently, it is the value that maximises the LOG-LIKELIHOOD $\ln L(\boldsymbol{\theta}|\mathbf{y})$, which is usually easier to compute. The MLE is usually denoted by $\hat{\boldsymbol{\theta}}$. If the observations are all independent and such that the density of the distribution generating y_i is $f_i(.;\boldsymbol{\theta})$, then their joint density is just $f(\mathbf{y};\boldsymbol{\theta}) = \prod_i f_i(y_i;\boldsymbol{\theta})$. In this case the log-likelihood is

$$\ln L(\boldsymbol{\theta}|\mathbf{y}) = \sum_{i=1}^{n} \ln f_i\left(y_i;\boldsymbol{\theta}\right) \ . \tag{7}$$

The likelihood function can be used to form confidence intervals, by finding the set of all values of $\boldsymbol{\theta}$ for which the likelihood (or, equivalently, the log-likelihood) exceeds some threshold.

Hypothesis testing can also be carried out in a likelihood-based framework. To test whether the data are consistent with an underlying value of $\boldsymbol{\theta}_0$, we can examine the LIKELIHOOD RATIO $\Lambda = L(\hat{\boldsymbol{\theta}}|\mathbf{y})/L(\boldsymbol{\theta}_0|\mathbf{y})$, or its logarithm. By definition of $\hat{\boldsymbol{\theta}}$, $L(\hat{\boldsymbol{\theta}}|\mathbf{y}) \geq L(\boldsymbol{\theta}_0|\mathbf{y})$. Values of Λ close to 1 (i.e. values of $\ln \Lambda$ close to zero) are consistent with the null hypothesis; larger values are not.

Likelihood-based procedures have a number of appealing properties. A precise statement is lengthy and theoretical — see, for example, Cox and Hinkley (1974) for a full discussion. For practical purposes however, the most important ones can be summarised as follows:

1. For many models based on the exponential family, MLEs have the smallest mean squared error of *any* estimator.

2. For such models, likelihood-based confidence intervals are generally the shortest that can be found, at a specified confidence level.

3. The most powerful test for distinguishing between two hypotheses is based on the likelihood ratio (the NEYMAN-PEARSON LEMMA). This means that if a weak signal is present in a noisy record, a likelihood ratio test may be able to detect it when other procedures cannot.

The only disadvantage to likelihood-based inference is that it requires the probability model $f(\mathbf{y};\boldsymbol{\theta})$ to be completely (and correctly!) specified. Results and conclusions will depend on this specification, so we need to ensure that the model structure is realistic. For example, if the observations arise as a time series then they are likely to be dependent so that **(7)** is incorrect. However, a standard factorisation of the joint density allows us to write the log-likelihood as

$$\ln L(\boldsymbol{\theta}|\mathbf{y}) = \sum_{i=1}^{n} \ln f_i\left(y_i|\mathcal{H}_i;\boldsymbol{\theta}_i\right) \ ,$$

where here $f_i\left(y_i|\mathcal{H}_i;\boldsymbol{\theta}_i\right)$ denotes the density of the ith observation given its history $\mathcal{H}_i$ say. This has the same form as **(7)**, and hence we can proceed as usual providing the history is

adequately accounted for within the model. It is this result that motivates the inclusion of previous days' values into the GLMs for daily climate time series.

In GLMs, where the observations all come from distributions within the exponential family with a common dispersion parameter ϕ, the log-likelihood for $\boldsymbol{\beta}$ and ϕ given n independent observations is, from (5) and (7),

$$\ln L\left(\boldsymbol{\beta}, \phi|\mathbf{y}\right) = \sum_{i=1}^{n} \frac{y_i\psi_i - b(\psi_i)}{a(\phi)} + c(y, \phi) \ . \tag{8}$$

The coefficient vector $\boldsymbol{\beta}$ enters the right-hand side here through the ψ_i terms, as described in the previous section.

In well-behaved problems, the value of $\boldsymbol{\beta}$ maximising the log-likelihood satisfies the p SCORE EQUATIONS

$$\frac{\partial \ln L}{\partial \beta_j} = \frac{1}{a(\phi)} \sum_{i=1}^{n} \frac{\partial}{\partial \beta_j} \left[y_i\psi_i - b(\psi_i)\right] \ (= U_j, \text{ say}) \ = 0 \qquad (j = 1, \ldots, p) \ , \tag{9}$$

whose solution clearly does not depend on ϕ.

B.1 Likelihood ratio tests

In the previous section, we noted that hypothesis testing can be carried out using likelihood ratios. We now summarise the procedure. Specifically, we suppose that the linear predictors in (4) have the form

$$\eta_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} \ ,$$

and we wish to test the null hypothesis $H_0 : \beta_{q+1} = \beta_{q+2} = \ldots = \beta_p = 0$, for some $q < p$. The likelihood ratio test procedure in this case is:

1. Fit the REDUCED MODEL (i.e. the model containing the first q predictors) using Maximum Likelihood; denote the resulting log-likelihood by $\ln L_0$.

2. Fit the model containing all of the xs, and denote the resulting log-likelihood by $\ln L_1$. This will never be less than $\ln L_0$.

3. Calculate the likelihood ratio test statistic $2\ln\Lambda = 2\left(\ln L_1 - \ln L_0\right)$. If this is larger than the appropriate percentage point of a χ^2 distribution with $(p - q)$ degrees of freedom, reject the null hypothesis; otherwise accept it.

There is a potential complication here for GLMs that involve an unknown dispersion parameter ϕ. This is because the log-likelihood (8) depends on ϕ, whence the likelihood ratio does also (although the $c(y, \phi)$ term cancels in the ratio). If ϕ is unknown, we can of course estimate it. However, in general we will obtain different estimates from each of

the models under consideration and this may affect our inference. Most standard software packages work around this problem by reporting the DEVIANCE (see below) rather than the log-likelihood. This software reports both. For models involving a dispersion parameter ϕ, the log-likelihoods are maximised with respect to both $\boldsymbol{\beta}$ and ϕ — thus the theory above is directly applicable.

B.2 Deviance

For the reasons given above, many software packages output deviances for GLMs rather than log-likelihoods. The deviance for a model is defined as

$$D = 2a(\phi)\left(\ln L_F - \ln L\right) \ ,$$

where L_F is the likelihood for a FULL MODEL in which we set $\mu_i = y_i$ for each i, and L is the likelihood for the model under consideration. Comparing this with **(8)**, we see that the deviance does not depend on ϕ.

The deviance is equivalent to the residual sum of squares in a linear regression (and is never negative) — in fact, for a GLM based on the normal distribution with constant variances, the deviance *is* the residual sum of squares. For this reason, procedures such as analysis of variance (which describes how different predictors in a linear regression account for the variability in the Ys) are generalised to 'analysis of deviance' in GLMs. F tests can be used to compare models, as in the standard regression case.

C Numerical algorithms for GLMs

C.1 Iterative weighted least squares

We now address the problem of calculating the maximum likelihood estimate for a GLM. In practice, the score equations **(9)** must be solved numerically in all but the simplest cases. Assembling all p equations into vector form, we seek the solution of

$$\mathbf{U}\left(\boldsymbol{\beta}\right) = \mathbf{0} \tag{10}$$

where $\mathbf{U}\left(\boldsymbol{\beta}\right) = \left(U_1, \ldots, U_p\right)'$ is the SCORE VECTOR of log-likelihood derivatives. $\mathbf{U}(.)$ is typically a nonlinear function of $\boldsymbol{\beta}$.

To solve equations of the form **(10)**, the Newton-Raphson algorithm may be used: start with an initial guess at the solution, $\boldsymbol{\beta}^{(0)}$ say, and then successively calculate

$$\boldsymbol{\beta}^{(t)} = \boldsymbol{\beta}^{(t-1)} - \left[\frac{\partial \mathbf{U}}{\partial \boldsymbol{\beta}}_{|\boldsymbol{\beta}^{(t-1)}}\right]^{-1} \mathbf{U}\left(\boldsymbol{\beta}^{(t-1)}\right) \tag{11}$$

until convergence is achieved. $\partial \mathbf{U}/\partial \boldsymbol{\beta}$ here is the $p \times p$ matrix of second derivatives of the log-likelihood with respect to $\boldsymbol{\beta}$. Note that, since the likelihood for a given $\boldsymbol{\beta}$ depends on the data, it should be regarded as the realised value of a random variable, as should its derivatives. Hence it is meaningful to consider the expected value of likelihood derivatives. In particular, the quantity $\mathbf{I}(\boldsymbol{\beta}) = -E_{\boldsymbol{\beta}}[\partial \mathbf{U}/\partial \boldsymbol{\beta}]$ is called the INFORMATION MATRIX for $\boldsymbol{\beta}$.

The reason for introducing this concept is that, when finding the maximum likelihood estimate of $\boldsymbol{\beta}$ in a GLM, it is common to replace the matrix $\partial \mathbf{U}/\partial \boldsymbol{\beta}$ in **(11)** by its expected value. The resulting maximisation algorithm is called the METHOD OF SCORING: the iterative scheme is

$$\boldsymbol{\beta}^{(t)} = \boldsymbol{\beta}^{(t-1)} + \left[\mathbf{I}\left(\boldsymbol{\beta}^{(t-1)}\right)\right]^{-1} \mathbf{U}\left(\boldsymbol{\beta}^{(t-1)}\right) \ . \tag{12}$$

With a log-likelihood of the form **(8)**, the derivative with respect to β_j is

$$U_j = \frac{\partial \ln L}{\partial \beta_j} = \frac{1}{a(\phi)} \sum_{i=1}^{n} \frac{\partial \ell_i}{\partial \beta_j} \ ,$$

where ℓ_i is a contribution from the ith observation. The chain rule gives

$$\frac{\partial \ell_i}{\partial \beta_j} = \frac{\partial \ell_i}{\partial \psi_i} \frac{\partial \psi_i}{\partial \mu_i} \frac{\partial \mu_i}{\partial \eta_i} \frac{\partial \eta_i}{\partial \beta_j} \ ,$$

where μ_i and η_i are the mean and linear predictor for the ith case (recall equation **(4)**). Dealing with each of these in turn, and referring to the properties of the exponential family on page 68:

1. $\partial \ell_i/\partial \psi_i = y_i - \partial b/\partial \psi_i = y_i - \mu_i$.
2. $\partial \psi_i/\partial \mu_i = [\partial \mu_i/\partial \psi_i]^{-1}$. Now $\mu_i = \partial b/\partial \psi_i$, so $\partial \mu_i/\partial \psi_i = \partial^2 b/\partial \psi_i^2$. But $\mathrm{Var}(Y_i) = a(\phi)\partial^2 b/\partial \psi_i^2$. Hence $\partial^2 b/\partial \psi_i^2 = \mathrm{Var}(Y_i)/a(\phi)$, and $\partial \psi_i/\partial \mu_i = a(\phi)/\mathrm{Var}(Y_i) = a(\phi)/V_i$, say.
3. $\partial \mu_i/\partial \eta_i$ depends on the particular link function used in equation **(4)**.
4. $\partial \eta_i/\partial \beta_j = x_{i,j}$ (the value of the jth covariate for the ith case).

Putting these results together and summing over all cases in the dataset, we find that the jth element of the score vector is given by

$$U_j = \sum_{i=1}^{n} \left[\frac{y_i - \mu_i}{V_i} \left(\frac{\partial \mu_i}{\partial \eta_i}\right) \left(\frac{\partial \eta_i}{\partial \beta_j}\right)\right] = \sum_{i=1}^{n} \left[\frac{y_i - \mu_i}{V_i} \left(\frac{\partial \mu_i}{\partial \eta_i}\right) x_{i,j}\right] \ . \tag{13}$$

Calculation of the information matrix $\mathbf{I}(\boldsymbol{\beta})$ is simplified by the fact that its (j,k)th element is equal to $E[U_j U_k]$ providing the underlying model is correct — in particular, providing the independence assumptions of the model hold. It can also be shown that the

scores all have mean zero, whence $\mathbf{I}(\boldsymbol{\beta})$ has an alternative representation as the covariance matrix of the score vector. These are standard results and will not be proved here (the former can be derived using straightforward algebra from representation **(13)** of the scores). However they enable us to deduce, after a little manipulation involving **(13)**, that if the model is correct the (j,k)th element of the information matrix is

$$\sum_{i=1}^{n}\left[\frac{1}{V_i}\left(\frac{\partial\eta_i}{\partial\beta_j}\right)\left(\frac{\partial\eta_i}{\partial\beta_k}\right)\left(\frac{\partial\mu_i}{\partial\eta_i}\right)^2\right]=\sum_{i=1}^{n}\left[\frac{x_{i,j}x_{i,k}}{V_i}\left(\frac{\partial\mu_i}{\partial\eta_i}\right)^2\right] \tag{14}$$

In matrix form then, we can write

$$\mathbf{I}\left(\boldsymbol{\beta}\right)=\mathbf{X}'\mathbf{W}\mathbf{X}\ , \tag{15}$$

where $\mathbf{W}$ is a diagonal $n\times n$ matrix with elements $w_{ii}=\left(\partial\mu_i/\partial\eta_i\right)^2/V_i$ (for computational purposes, in fact it is convenient to use $a(\phi)\partial\mu_i/\partial\psi_i$ in place of V_i here, since then $a(\phi)$ appears as a constant that can be omitted from the iterative scheme **(17)** below). All of these quantities are functions of $\boldsymbol{\beta}$.

This matrix representation can be used in the iterative scoring algorithm **(12)**: multiplying both sides of that equation by $\mathbf{I}\left(\boldsymbol{\beta}^{(t-1)}\right)$ and substituting **(15)** for $\mathbf{I}\left(\boldsymbol{\beta}^{(t-1)}\right)$, we find

$$\mathbf{X}'\mathbf{W}^{(t-1)}\mathbf{X}\boldsymbol{\beta}^{(t)}=\mathbf{X}'\mathbf{W}^{(t-1)}\mathbf{X}\boldsymbol{\beta}^{(t-1)}+\mathbf{U}\left(\boldsymbol{\beta}^{(t-1)}\right)\ . \tag{16}$$

Noting that $\mathbf{X}\boldsymbol{\beta}^{(t-1)}=\boldsymbol{\eta}^{(t-1)}$, the vector of linear predictors at iteration $t-1$, and that the score vector can itself be written as a vector product involving the matrix $\mathbf{X}'\mathbf{W}$ (from **(13)**), the scoring algorithm can finally be written in matrix form as

$$\left[\mathbf{X}'\mathbf{W}^{(t-1)}\mathbf{X}\right]\boldsymbol{\beta}^{(t)}=\mathbf{X}'\mathbf{W}^{(t-1)}\mathbf{z}^{(t-1)}\ , \tag{17}$$

where $\mathbf{z}^{(\mathbf{t-1})}$ is an $n\times 1$ vector whose ith element is

$$z_i^{(t-1)}=\eta_i^{(t-1)}+\left(y_i-\mu_i^{(t-1)}\right)\left(\frac{\partial\eta}{\partial\mu}_{|\mu_i^{(t-1)}}\right)\ .$$

Equation **(17)** in fact gives the solution of the weighted least-squares regression of $\mathbf{z}^{(t-1)}$ upon $\mathbf{X}$, with weights contained in the diagonal elements of $\mathbf{W}^{(t-1)}$. The need for iteration arises because $\mathbf{z}$ and $\mathbf{W}$ both depend, in general, upon $\boldsymbol{\beta}$. Expressed in this form, the algorithm for fitting GLMs is referred to as ITERATIVE WEIGHTED LEAST SQUARES (IWLS). An additional advantage is that for large samples, the covariance matrix of the final parameter estimates is $\left[\mathbf{X}'\mathbf{W}\mathbf{X}\right]^{-1}$, which emerges as a by-product of the fitting procedure. This can be used, for example, to derive standard errors for the parameter estimates. See Appendix C.2 below for more details on this.

C.1.1 The information matrix versus the Hessian

In what follows, it will be useful to understand the implications of replacing the Hessian matrix of the log-likelihood by its expected value in the algorithm above. Differentiating **(13)** with respect to β_k, we find that the (j,k)th element of the Hessian matrix is

$$\begin{aligned}\frac{\partial U_j}{\partial \beta_k} &= \sum_{i=1}^{n}\left\{(y_i-\mu_i)\frac{\partial}{\partial\beta_k}\left[\frac{x_{i,j}}{V_i}\left(\frac{\partial\mu_i}{\partial\eta_i}\right)\right]-\frac{\partial\mu_i}{\partial\beta_k}\left[\frac{x_{i,j}}{V_i}\left(\frac{\partial\mu_i}{\partial\eta_i}\right)\right]\right\}\\ &= \sum_{i=1}^{n}\left\{(y_i-\mu_i)\frac{\partial}{\partial\beta_k}\left[\frac{x_{i,j}}{V_i}\left(\frac{\partial\mu_i}{\partial\eta_i}\right)\right]-\frac{x_{i,j}x_{i,k}}{V_i}\left(\frac{\partial\mu_i}{\partial\eta_i}\right)^2\right\}\ . \end{aligned} \tag{18}$$

Note that, if we take expectations, the first term here vanishes and the expression reduces (with a change of sign), to the (j,k)th element of the information matrix at **(14)**. The first term will also vanish if the term in square brackets is invariant with respect to β_k, since then the derivative will be identically zero. If this is the case then the Hessian matrix will always be exactly equal to its expected value. Such invariance will occur when $\partial\mu_i/\partial\eta_i \propto V_i$ i.e. for a particular choice of the link function $g(.)$ in **(4)**. This choice is called the CANONICAL LINK. From item 2 in the list preceding **(13)** (page 75), we see that V_i may be defined as $a(\phi)\times\partial\mu_i/\partial\psi_i$. Hence the canonical link function for any GLM is $g(\mu_i)=\psi(\mu_i)$. Refer to Table 8 for the canonical links $\psi(.)$ in some standard distributions.

If the information is *not* equal to the negative Hessian matrix, it is natural to question the applicability of the scoring algorithm. We now give a heuristic justification for its use. From **(11)** and **(12)**, it is clear that the algorithm will provide a good approximation to the Newton-Raphson iterative scheme (and hence should work in well-behaved problems) providing $[\partial\mathbf{U}/\partial\boldsymbol{\beta}]^{-1}+\mathbf{I}^{-1}(\boldsymbol{\beta})$ is small. Now, given n independent observations, the log-likelihood and its derivatives are sums of n terms and therefore deviate from their expectations by quantities that are $O_p\left(n^{1/2}\right)$. Hence, if the model is correct then in some neighbourhood of the true parameter vector we can write

$$-\frac{\partial\mathbf{U}}{\partial\boldsymbol{\beta}}=\mathbf{I}(\boldsymbol{\beta})+\mathbf{E}\ , \tag{19}$$

say, where $\mathbf{E}$ is a matrix whose elements are $O_p\left(n^{1/2}\right)$. Writing $\mathbf{1}$ for an identity matrix, we therefore have

$$-\left[\frac{\partial\mathbf{U}}{\partial\boldsymbol{\beta}}\right]^{-1}=\mathbf{I}^{-1}(\boldsymbol{\beta})\left[\mathbf{1}+\mathbf{I}^{-1}(\boldsymbol{\beta})\mathbf{E}\right]^{-1}\ .$$

Now the elements of $\mathbf{I}^{-1}(\boldsymbol{\beta})$ are $O_p\left(n^{-1}\right)$, so those of $\mathbf{I}^{-1}(\boldsymbol{\beta})\mathbf{E}$ are $O_p\left(n^{-1/2}\right)$. Hence, for large n we have

$$-\left[\frac{\partial\mathbf{U}}{\partial\boldsymbol{\beta}}\right]^{-1}\approx\mathbf{I}^{-1}(\boldsymbol{\beta})\left[\mathbf{1}-\mathbf{I}^{-1}(\boldsymbol{\beta})\mathbf{E}\right]=\mathbf{I}^{-1}(\boldsymbol{\beta})+\mathbf{M}\ ,$$

where now $\mathbf{M}$ is a matrix whose elements are $O_p\left(n^{-3/2}\right)$ (in contrast with those of $\mathbf{I}^{-1}(\boldsymbol{\beta})$, which are $O_p\left(n^{-1}\right)$). Hence, for sufficiently large samples, replacing $\partial\mathbf{U}/\partial\boldsymbol{\beta}$ with its expected

value should not cause numerical problems *providing* the search is restricted to an appropriate neighbourhood of the true parameter vector. This proviso arises because in **(19)**, $\mathbf{I}(\boldsymbol{\beta})$ is an expected value computed as though $\boldsymbol{\beta}$ is the true parameter vector. If $\mathbf{I}(\boldsymbol{\beta})$ varies substantially with $\boldsymbol{\beta}$ then we may expect problems away from the true value (or equivalently, away from the MLE, since this also deviates from the true value by $O_p(n^{1/2})$).

C.1.2 Normal-heteroscedastic models

The normal-heteroscedastic family of models, described in Section 3.1, does not strictly fall within the framework outlined above. However, as described in Chandler (2005) and Yang et al. (2005), these models can be fitted by noting that if $Y_i \sim N(\mu_i, \sigma_i^2)$ then the squared residual $e_i^2 = (Y_i - \mu - i)^2$ has a distribution proportional to chi-squared on one degree of freedom; which is the same as a gamma distribution with mean σ_i^2 and shape parameter $1/2$. The fitting algorithm for normal-heteroscedastic models is therefore as follows:

1. Fit the mean component of the model (defined by equation 1 on page 8) to the observations Y_i using least squares.
2. Compute the squared residuals $\{e_i^2\}$ from the mean component of the model, and fit the dispersion component (defined by equation 2) to these squared residuals as a gamma GLM.
3. Refit the mean component of the model using weighted least squares, where the weights for each case are the inverse of the fitted values from the dispersion component of the model. Go back to step 2, and iterate to convergence.

The theory outlined elsewhere in this appendix needs to be modified slightly to allow for the reweighting in step 3 of this algorithm, but conceptually the modifications are straightforward. They are omitted here, to avoid complicating the mathematical presentation.

C.2 Covariance matrix of the estimates

Having demonstrated how maximum likelihood parameter estimates may be obtained, we now consider their covariance matrix. From **(17)**, it is clear that the MLE satisfies

$$\hat{\boldsymbol{\beta}} = [\mathbf{X}'\mathbf{W}\mathbf{X}]^{-1}\,\mathbf{X}'\mathbf{W}\mathbf{z}\ , \tag{20}$$

where all quantities are evaluated at $\hat{\boldsymbol{\beta}}$. The required covariance matrix may be estimated by considering the covariance of the right hand side here as a function of $\boldsymbol{\beta}$, and evaluating it at $\hat{\boldsymbol{\beta}}$. Note that as a function of $\boldsymbol{\beta}$, $[\mathbf{X}'\mathbf{W}\mathbf{X}]^{-1}$ is non-random. Note also the standard result that if $\mathbf{Y}$ is a vector of random variables and $\mathbf{A}$ a matrix such that $\mathbf{A}\mathbf{Y}$ is defined, $\mathrm{Var}(\mathbf{A}\mathbf{Y}) = \mathbf{A}\mathrm{Var}(Y)\mathbf{A}'$. Hence the covariance matrix of the right hand side of **(20)** is

$$[\mathbf{X}'\mathbf{W}\mathbf{X}]^{-1}\,\mathrm{Var}\,(\mathbf{X}'\mathbf{W}\mathbf{z})\,[\mathbf{X}'\mathbf{W}\mathbf{X}]^{-1}\ , \tag{21}$$

since $[\mathbf{X}'\mathbf{W}\mathbf{X}]^{-1}$ is a symmetric matrix.

The next step is to note that the covariance matrix of $\mathbf{X}'\mathbf{W}\mathbf{z}$ is, for fixed $\boldsymbol{\beta}$, the same as that of the score vector $\mathbf{U}$. To see this, equate the right hand sides of equations **(16)** and **(17)** to obtain $\mathbf{X}'\mathbf{W}\mathbf{X}\boldsymbol{\beta} + \mathbf{U} = \mathbf{X}'\mathbf{W}\mathbf{z}$ — this relationship defines $\mathbf{z}$ for any $\boldsymbol{\beta}$. But for fixed $\boldsymbol{\beta}$, $\mathbf{X}'\mathbf{W}\mathbf{X}\boldsymbol{\beta}$ is non-random, and hence can be ignored in variance calculations. The estimated covariance matrix of $\hat{\boldsymbol{\beta}}$ is therefore

$$\hat{\mathrm{Var}}\left(\hat{\boldsymbol{\beta}}\right) = [\mathbf{X}'\mathbf{W}\mathbf{X}]^{-1}\,\mathrm{Var}\,(\mathbf{U})\,[\mathbf{X}'\mathbf{W}\mathbf{X}]^{-1} \quad . \tag{22}$$

If observations are independent, we have seen (page 76) that $\mathrm{Var}\,(\mathbf{U}) = \mathbf{X}'\mathbf{W}\mathbf{X}$. In this case, **(22)** reduces to $[\mathbf{X}'\mathbf{W}\mathbf{X}]^{-1}$. This is the 'default' formula used by the software to calculate standard errors, if no spatial dependence structure is specified for a given model.

More frequently however, in climatological applications observations are obtained from a network of sites. In this case, inter-site dependence means that observations from different sites on the same day cannot be regarded as independent; however, in general there is nothing to stop us from estimating $\boldsymbol{\beta}$ by maximising the 'independence' log-likelihood. In this case, tests based on likelihood ratios and deviance must be modified to account for the inter-site dependence — these modifications can be complex. However, tests based on the covariance matrix **(22)** can be modified straightforwardly, providing we can find an easily computable estimate of $\mathrm{Var}\,(\mathbf{U})$. Since $\mathbf{U}$ is a sum over all observations in the database, we can write

$$\mathbf{U} = \sum_{t=1}^{T}\sum_{s=1}^{S_t}\mathbf{U}_{st} \ ,$$

where $\mathbf{U}_{st}$ denotes the contribution from site s on day t. Hence

$$\mathrm{Var}\,(\mathbf{U}) = \mathrm{Var}\left(\sum_{t=1}^{T}\sum_{s=1}^{S_t}\mathbf{U}_{st}\right) = \sum_{t=1}^{T}\mathrm{Var}\left(\sum_{s=1}^{S_t}\mathbf{U}_{st}\right) \ ,$$

assuming that score contributions from different days are uncorrelated (which will be the case so long as the model contains an adequate representation of temporal dependence — see Appendix B above, and also note that from equation **(13)**, contributions to the score vector are essentially weighted residuals from the fitted model). Now, since each contribution to the score vector has expected value zero (page 76), we have

$$\sum_{t=1}^{T}\mathrm{Var}\left(\sum_{s=1}^{S_t}\mathbf{U}_{st}\right) = \sum_{t=1}^{T}\mathrm{E}\left[\left(\sum_{s=1}^{S_t}\mathbf{U}_{st}\right)\left(\sum_{s=1}^{S_t}\mathbf{U}_{st}\right)'\right] \ ,$$

where $\mathrm{E}\,[.]$ denotes expectation. For large T, we can use the observed values of the square-bracketed terms to estimate their expectations, and obtain the estimator

$$\hat{\mathrm{Var}}\,(\mathbf{U}) = \sum_{t=1}^{T}\left(\sum_{s=1}^{S_t}\mathbf{U}_{st}\right)\left(\sum_{s=1}^{S_t}\mathbf{U}_{st}\right)' \ , \tag{23}$$

which is easily computable at little extra cost, once $\boldsymbol{\beta}$ has been found. The software uses this estimator of $\mathrm{Var}\,(\mathbf{U})$, in conjunction with **(22)**, to compute standard errors whenever the user specifies a spatial dependence structure in a model.

C.3 Dependence-adjusted likelihood ratio

Appendix B.1 above presented the likelihood ratio test procedure for discriminating between two nested models. The theory underlying this procedure assumes that the observations are conditionally independent given the covariates. Specifically, it relies upon the standard identity

$$\mathrm{Var}\left[\mathbf{U}\left(\boldsymbol{\beta}_0\right)\right] = -\mathrm{E}\left[\mathbf{H}\left(\boldsymbol{\beta}_0\right)\right] \ ,$$

where $\boldsymbol{\beta}_0$ is the underlying true value of $\boldsymbol{\beta}$ (this assumes, of course, that the model is correctly specified so that it is meaningful to speak of such a 'true' value) and $\mathbf{H}$ is the Hessian matrix of second derivatives of the 'independence' log-likelihood function **(8)**. Inter-site dependence affects the covariance matrix of $\mathbf{U}\left(\boldsymbol{\beta}_0\right)$, as detailed above; however, the Hessian of the independence log-likelihood is unchanged.

This observation has been used by Chandler and Bate (2007) to develop an adjustment to the likelihood ratio test in the presence of inter-site dependence. The idea is to construct a modified inference function that is maximised at the 'independence' MLE $\hat{\boldsymbol{\beta}}$, but with Hessian $-\mathcal{R}^{-1}$, where $\mathcal{R}$ is the 'robust' covariance estimate obtained by combining **(23)** with **(22)**. Denoting by ℓ_{IND} the 'independence' log-likelihood **(8)**, this modified inference function is defined as

$$\ell_{ADJ}(\boldsymbol{\beta}) = \ell_{IND}\left(\boldsymbol{\beta}^*\right) \tag{24}$$

for a linear transformation

$$\boldsymbol{\beta}^* = \hat{\boldsymbol{\beta}} + \mathbf{A}\left(\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}\right). \tag{25}$$

By equating Taylor series expansions about $\hat{\boldsymbol{\beta}}$, it can be shown that the matrix $\mathbf{A}$ satisfies

$$\mathbf{A} = \left\{\left[\mathcal{N}^{-1}\right]^{1/2}\right\}^{-1}\left[\mathcal{R}^{-1}\right]^{1/2} \ , \tag{26}$$

where $\mathcal{N}$ is the 'naive' covariance matrix $\left[\mathbf{X}'\mathbf{W}\mathbf{X}\right]^{-1}$. The matrix square roots in **(26)** are not uniquely defined; however, the second-order Taylor expansion underlying the adjustment *is* unique. The software uses Cholesky square roots.

Notice that the adjustment here is model-dependent, and hence cannot be used for 'global' correction of the log-likelihood. It can be used, however, to compare two models when one is a special case of the other. In this case, the missing terms in the simpler model can be treated as having zero coefficients and the adjusted log-likelihood $\ell_{ADJ}(\boldsymbol{\beta})$ can be treated as though it is the true log-likelihood function. For example, a likelihood ratio test can be performed by maximising $\ell_{ADJ}(\boldsymbol{\beta})$ under both the reduced and full models, to obtain estimates $\hat{\boldsymbol{\beta}}_0$ and $\hat{\boldsymbol{\beta}}_1$, say: then the adjusted likelihood ratio statistic

$$2\left[\ell_{ADJ}(\hat{\boldsymbol{\beta}}_1) - \ell_{ADJ}(\hat{\boldsymbol{\beta}}_0)\right] \tag{27}$$

can be compared with percentage points of the appropriate χ^2 distribution in the usual way. To maximise $\ell_{ADJ}(\boldsymbol{\beta})$ under the full model is straightforward: it is achieved by maximising the 'independence' log-likelihood using the usual algorithms. Maximisation under the

reduced model is more difficult; moreover, in typical applications the reduced model will already have been fitted using the 'independence' log-likelihood (to yield an estimate $\tilde{\boldsymbol{\beta}}_0$, say), and the user may wish to use this existing estimate rather than carry out an extra, computationally expensive maximisation. To compute **(27)** it is therefore tempting to consider using $\tilde{\boldsymbol{\beta}}_0$ in place of $\hat{\boldsymbol{\beta}}_0$. However, since by definition $\ell_{ADJ}\left(\hat{\boldsymbol{\beta}}_0\right) \geq \ell_{ADJ}\left(\tilde{\boldsymbol{\beta}}_0\right)$, such a procedure will inflate the values of the test statistic. This inflation can be corrected using a secondary adjustment: specifically, an alternative to **(27)** is

$$2c\left[\ell_{ADJ}(\hat{\boldsymbol{\beta}}_1) - \ell_{ADJ}(\tilde{\boldsymbol{\beta}}_0)\right] \quad , \tag{28}$$

for some scaling factor $c \in (0, 1)$. The value of c can be computed explicitly if (a) $\ell_{ADJ}(\cdot)$ is quadratic in the neighbourhood of interest (b) the reduced model is defined in terms of a linear constraint on the parameters of the full model. In the applications for which this software is designed, both criteria are likely to be satisfied.[15]

C.4 Nonlinear transformations of the covariates

The above theory provides an algorithm for finding the MLE of the parameter vector $\boldsymbol{\beta}$ in a model of the form **(4)**. In some situations, however, we may wish to represent one or more covariates themselves as nonlinear transformations of some underlying quantity. For example, some of the 'year' effects in Table 3 involve parameters that are, in general unknown. Similarly, there are unknown parameters involved in some of the weighting schemes in Table 5. The general setup here is that $x_{i,j}$ can be written as $f(x^*_{i,j}; \theta)$ where $x^*_{i,j}$ is the value of the underlying quantity for the ith case in the dataset, and $f(.;.)$ is a function of known parametric form. This scenario is non-standard within GLMs. For simplicity, to start with we assume that $\boldsymbol{\theta}$ is a scalar and write θ.

Although the estimation of nonlinear transformations is nonstandard, the MLE of θ can in principle be found via a straightforward extension of the IWLS algorithm above. The key point to note is the presence, before simplification in equations **(13)** and **(14)**, of the quantities $\partial\eta_i/\partial\beta_j$ and $\partial\eta_i/\partial\beta_k$. In the standard GLM case, these partial derivatives are equal to $x_{i,j}$ and $x_{j,k}$ respectively — it is the fact that these are elements of the matrix $\mathbf{X}$ that enables us to write the scoring algorithm in the IWLS form **(17)**. Following the previous argument up to **(13)**, the score for θ is just

$$U_\theta = \sum_{i=1}^{n}\left[\frac{y_i - \mu_i}{V_i}\left(\frac{\partial\mu_i}{\partial\eta_i}\right)\left(\frac{\partial\eta_i}{\partial\theta}\right)\right] \quad , \tag{29}$$

and the information matrix for the augmented parameter vector $(\boldsymbol{\beta}\ \theta)'$ can be constructed as before via covariances of the scores. A matrix representation of the resulting scoring

[15] In fact, in Rglimclim the `anova` method for fitted model objects uses the 'vertical' adjustment of the likelihood ratio statistic from Chandler and Bate (2007) rather than the 'horizontal' adjustment described above. The vertical version is easier to compute, and is also slightly preferable on theoretical grounds.

algorithm can therefore be obtained as

$$\mathbf{X}^{*\prime}\mathbf{W}^{(t-1)}\mathbf{X}^{*}\begin{pmatrix}\boldsymbol{\beta}\\ \theta\end{pmatrix}^{(t)} = \mathbf{X}^{*\prime}\mathbf{W}^{(t-1)}\mathbf{z}^{(t-1)} \ , \tag{30}$$

where $\mathbf{X}^{*}$ is now the $n \times (p+1)$ matrix obtained by adding an extra column to $\mathbf{X}$, whose elements are the values of $\partial\eta/\partial\theta$ for each case in the dataset. By analogy with **(16)**, the definition of $\mathbf{z}^{(t-1)}$ must also change slightly — its ith element is now

$$z_i^{(t-1)} = \eta_i^{(t-1)} + \theta^{(t-1)}\frac{\partial\eta_i^{(t-1)}}{\partial\theta^{(t-1)}} + \left(y_i - \mu_i^{(t-1)}\right)\left(\frac{\partial\eta}{\partial\mu}_{|\mu_i^{(t-1)}}\right) \ .$$

Conceptually at least then, the extension of the usual IWLS algorithm to deal with nonlinear transformations of covariates is straightforward: simply augment the original p covariates with a set of 'dummy' covariates corresponding to derivatives of the linear predictor with respect the parameters of interest, and proceed as normal. Of course, since some elements of $\mathbf{X}$ now depend on θ, they will need to be updated after each iteration — this increases the computational load somewhat, particularly for large datasets. A further consequence is that the information matrix $\mathbf{X}^{*\prime}\mathbf{W}^{(t-1)}\mathbf{X}^{*}$ may vary rapidly for some nonlinear parameterisations. In this case, from the discussion in the previous section, we may expect convergence problems with the scoring algorithm.

It turns out that a small modification of the algorithm will guarantee convergence to a maximum. To introduce this, let $\ell(.)$ be an arbitrary log-likelihood function for a parameter vector $\boldsymbol{\theta}$, and let $\mathbf{U}(.)$ be the corresponding score vector. Then, providing $\ell(.)$ is continuous and differentiable in the neighbourhood of $\boldsymbol{\theta}$, there exists a radius δ (which may be small) such that, for all vectors $\boldsymbol{\epsilon}$ with $|\boldsymbol{\epsilon}| < \delta$,

$$\ell\left(\boldsymbol{\theta}+\boldsymbol{\epsilon}\right) = \ell\left(\boldsymbol{\theta}\right) + \lambda\boldsymbol{\epsilon}'\mathbf{U}\left(\boldsymbol{\theta}\right)$$

for some $\lambda > 0$. In particular, this holds if we set $\boldsymbol{\epsilon} = \mathbf{D}\mathbf{U}\left(\boldsymbol{\theta}\right)$ where $\mathbf{D}$ is a sufficiently small, symmetric, positive definite matrix. In this case,

$$\ell\left(\boldsymbol{\theta}+\boldsymbol{\epsilon}\right) = \ell\left(\boldsymbol{\theta}\right) + \lambda\mathbf{U}'\left(\boldsymbol{\theta}\right)\mathbf{D}\mathbf{U}\left(\boldsymbol{\theta}\right) \ ,$$

which cannot be less than $\ell\left(\boldsymbol{\theta}\right)$ since $\lambda > 0$ and $\mathbf{D}$ is positive definite.

The relevance of this result is as follows: a step of the unmodified scoring algorithm changes the current value of the parameter vector by $\left[\mathbf{X}^{*\prime}\mathbf{W}\mathbf{X}^{*}\right]^{-1}\mathbf{U}\left(\boldsymbol{\theta}\right)$, in an obvious notation (compare with **(16)** to verify this). This is of the form $\mathbf{D}\mathbf{U}\left(\boldsymbol{\theta}\right)$, where $\mathbf{D}$ is symmetric and positive definite. Therefore, from the result above there exists a $\rho > 0$ such that changing the current value of the parameter vector by $\rho\left[\mathbf{X}^{*\prime}\mathbf{W}\mathbf{X}^{*}\right]^{-1}\mathbf{U}\left(\boldsymbol{\theta}\right)$ is guaranteed to increase the log-likelihood. The implication is that, if the log-likelihood decreases during any iteration of the unmodified algorithm, the step size was too large and we should reduce it.

To stabilise the algorithm therefore, if any step reduces the log-likelihood we successively reduce the step size until an increase is obtained. The literature recommends the use of rather

simple reduction schemes (e.g. successively halving the step size). However, experience shows that in particularly ill-behaved problems such schemes can require many successive reductions before an increase is found. We therefore halve the step size once and then try and locate the optimal step size by fitting a quadratic to the log-likelihood surface along the search direction. This modification guarantees eventual convergence to a maximum of the likelihood surface, although when nonlinear parametrisations are involved this maximum may be local rather than global (a trivial example of this, which can be overcome by suitable reparametrisation, arises when estimating the phase of a long-term cycle with a fixed frequency — the likelihood is then periodic with respect to the phase).

C.4.1 Miscellaneous details

A couple of points are worth noting with respect to this algorithm, and its use in the software provided here:

1. $\partial\eta/\partial\theta$ contains contributions not just from the underlying covariate itself, but also from interactions with other covariates. In most cases, this is straightforward. Care needs to be taken, however, if two interacting covariates (the jth and kth, say) are both transformations involving the *same* parameter θ. In this case (which may not always correspond to a sensible model!) the contribution to $\partial\eta/\partial\theta$ from the interaction term $x_{i,j}x_{i,k}$ is
$$x_{i,j}\frac{\partial x_{i,k}}{\partial\theta} + x_{i,k}\frac{\partial x_{i,j}}{\partial\theta} \ .$$
The software considers contributions $\partial\eta/\partial\theta$ involving the derivatives of each covariate in turn. In the example above, the contribution involving $\partial x_{i,j}/\partial\theta$ will pick up the second term, while that involving $\partial x_{i,k}/\partial\theta$ will pick up the first. One of those magical mathematical fortuities that brightens up a programmer's life.

2. When estimating parameters in schemes for computing weighted averages (Table 5), the weight attached to site r when computing an averages for site s is of the form
$$w_{r,s} = \left(w^*_{r,s}\right) / \left(\sum_j w^*_{j,s}\right) \ , \tag{31}$$
where the sum is over all sites contributing to the average. Here, $w^*_{j,s}$ is the non-normalised weight, whose functional form is known. The resulting covariate then takes the form
$$\sum_r w_{r,s} f\left(Y^{(r)}_{t-k}\right)$$
for the 'average of transformed values' case (codes 11–99 in Table 4), and
$$f\left(\sum_r w_{r,s} Y^{(r)}_{t-k}\right)$$

for the 'transformation of averages' case (codes 111–199). For the two different cases, the contributions to $\partial\eta/\partial\eta$ are

$$\sum_r \frac{\partial w_{r,s}}{\partial\theta} f\left(Y_{t-k}^{(r)}\right) \quad \text{and} \quad f'\left(\sum_r \frac{\partial w_{r,s}}{\partial\theta} Y_{t-k}^{(r)}\right) \sum_r \frac{\partial w_{r,s}}{\partial\theta} Y_{t-k}^{(r)} ,$$

respectively. Both expressions require $\partial w_{r,s}/\partial\theta$ which, from **(31)**, is given by

$$\frac{\partial w_{r,s}}{\partial\theta} = \left[\sum_j w_{j,s}^*\right]^{-2} \left(\frac{\partial w_{r,s}^*}{\partial\theta} \sum_j w_{j,s}^* - w_{r,s}^* \sum_j \frac{\partial w_{j,s}^*}{\partial\theta}\right) .$$

Note, incidentally, that if $f(.)$ here is an indicator function (e.g. zero if its argument is 0, 1 otherwise), $f'(.)$ is zero almost everywhere so that parameters in weighting schemes for the 'transformation of averages' case cannot be estimated. This makes perfect sense — the covariate value will be 1 or 0 regardless of the weighting scheme used, so in this case there is no information in the data regarding the parameters.

D Residuals

The model fitting software generates a variety of residual analyses by default. Residuals can be used to check both the systematic component of a model, and its distributional assumptions. They can also be used in simulation (see Appendix E below).

D.1 Types of residual

There is no unique definition of a 'residual' for a GLM. The main residual measures used in this software are as follows:

PEARSON RESIDUALS: these are defined in such a way that, if the model is correct, they all come from distributions with zero mean and the same variance. They can be defined as

$$r_i^{(P)} = K\frac{Y_i - \mu_i}{\sigma_i} , \tag{32}$$

where μ_i and σ_i are the modelled mean and standard deviation for the ith case in the dataset and K is a constant that may be chosen to make the residuals as interpretable as possible. Often it will be sensible to set $K = 1$, so that the residuals all come from distributions with zero mean and unit variance. For a gamma GLM, however, we have $\sigma_i = \mu_i/\sqrt{\nu}$, where ν is the common shape parameter of the distributions (see Appendix A). In this case, putting $K = 1/\sqrt{\nu}$ gives the residual measure $(Y_i - \mu_i)/\mu_i$, which is just the proportional error in prediction. This might be preferred as being more directly interpretable.

ANSCOMBE RESIDUALS: These are model residuals defined, in some sense, to have a distribution that is as close to Gaussian as possible. This is extremely useful for simulation of dependent sequences at several sites (see Appendix E below), particularly when the variable of interest is continuous. For a gamma GLM, the Anscombe residual for the ith case is defined to be $(y_i/\mu_i)^{1/3}$.

D.2 Checking the distributional assumptions

For continuous response variables, the easiest way to check the form of the forecast distribution is via quantile-quantile plots of suitably-defined residuals. 'Suitably-defined' here means that if the model is correct, all residuals come from identical distributions. For normal distributions, this is easily achieved: if $Y_i \sim N(\mu_i, \sigma_i^2)$ then $Z_i = (Y_i - \mu_i)/\sigma_i$ has a standard normal for all i. For the gamma family of distributions, the quantities Y_i/μ_i all have the same gamma distribution under the model.

Checks for discrete variables are more difficult. Here we focus on the binary case, and consider specifically the modelling of rainfall occurrence. If we collect together all of the days when the forecast probability of rain is close to some preassigned value p^*, then the overall proportion of these days upon which rainfall was actually observed should be close to p^*. An overview of the ideas is given by Dawid (1986). For practical implementation, we collect together groups of days for which forecast probabilities are in the intervals $(0.0, 0.1), (0.1, 0.2), \ldots, (0.9, 1.0)$ and compute observed and expected proportions of rainy days within each of these groups. Unless there is agreement across the whole range of forecast probabilities, there is something wrong with the probability structure of the model.

D.3 Checking systematic structure

Traditionally in regression modelling, the systematic structure of a model is checked by plotting residuals against predicted values, and against the covariates themselves. Such plots may be produced both for predictors which appear in the model, and for potential covariates that may need to be accounted for. Any apparent structure in these plots indicates a problem with the model. However, in many climate datasets such plots contain too many data points to distinguish any structure. For this reason, rather than plotting individual residuals we focus on summary statistics for residual measures over subgroups of observations. The software computes mean Pearson residuals for each month, year and site. This allows the modeller to check very quickly that the model captures the seasonal and regional structure in the data, along with any trends.

To aid the interpretation of such mean residuals, it is helpful to calculate their standard errors, which can be converted into confidence bands if desired. Suppose that all residuals are expected to have mean μ_ε and variance σ_ε^2 under the model, and a mean residual ($\overline{r}$, say) is computed over a large subset of M cases. Then 95% limits for this mean are at $\mu_\epsilon \pm 1.96$s.e.$(\overline{r})$, where s.e.$(\overline{r})$ is the standard error of the mean residual under the model.

This standard error can be calculated as follows: the mean residual is defined as

$$\overline{r} = \frac{1}{M} \sum_{t=1}^{T} \sum_{s=1}^{S} \chi_{ts} r_{ts} \ , \tag{33}$$

where T is the number of days in the subset under consideration; S the number of sites; χ_{ts} is an indicator taking the value 1 if a residual is available for site s on day t, 0 otherwise; and r_{ts} is the residual at site s on day t. Since M is the total number of observations in the subset, we have $M = \sum_{t=1}^{T} \sum_{s=1}^{S} \chi_{ts}$. When there is dependence between residuals at different sites on the same day (but independence between days), denote by $c_{s_1 s_2}$ the correlation between residuals at sites s_1 and s_2 on the same day. Then we have

$$\begin{aligned} \mathrm{Var}(\overline{r}) = \frac{1}{M^2} \sum_{t=1}^{T} \sum_{s_1=1}^{S} \sum_{s_2=1}^{S} \chi_{ts_1} \chi_{ts_2} \mathrm{cov}(r_{ts_1}, r_{ts_2}) &= \frac{\sigma_\varepsilon^2}{M^2} \sum_{s_1=1}^{S} \sum_{s_2=1}^{S} \sum_{t=1}^{T} \chi_{ts_1} \chi_{ts_2} c_{s_1 s_2} \\ &= \frac{\sigma_\varepsilon^2}{M^2} \sum_{s_1=1}^{S} \sum_{s_2=1}^{S} n_{s_1 s_2} c_{s_1 s_2} \ , \end{aligned} \tag{34}$$

where $n_{s_1 s_2}$ is the number of days for which sites s_1 and s_2 both have data. Taking the square root of **(34)** now yields the required standard error.

In implementing **(34)**, the software calculates a single set of inter-site correlations $\{c_{s_1 s_2}\}$ from the entire dataset, and applies these to all subsets.

E Simulation

This section gives an overview of the algorithms used in the software for simulation of daily sequences using the fitted GLMs.

E.1 Random number generator

The simulation program makes extensive use of pseudo-random numbers. Rather rely on the adequacy of generators 'built-in' to FORTRAN compilers (which are generally inadequately documented, and occasionally use rather poor algorithms), the routines used here are based on a uniform random number generator with excellent properties (Marsaglia and Zaman, 1991). These routines were written by myself together with Paul Northrop, and are suitable for extensive simulation excercises. The random number generation code is included with the software distribution. Full details, including documentation and references, are available from http://www.homepages.ucl.ac.uk/~ucakarc/work/randgen.html.

E.2 Spatial dependence — continuous variables

To generate realistic sequences of simulated climate data at several sites simultaneously, it is necessary to account adequately for the dependence between sites. The literature offers a plethora of possible methods for generating such dependent sequences. This software offers a limited, but hopefully adequate, range of options. All of these options are specified in such a way that, given observations at some sites on a particular day, the conditional distribution for missing observations at other sites can be calculated. This allows missing data to be imputed.

For continuous random variables, a convenient way of generating dependent data is to generate a multivariate normal random vector using a standard algorithm, and then transform the resulting values so that they can be regarded as coming from the correct marginal distributions. In this way, spatial dependence is completely characterised by the correlation structure of the multivariate normal distribution. Furthermore, if some elements of a multivariate normal random vector have been observed then a standard result enables us to compute the conditional distribution of the remaining elements (which is still multivariate normal) — this enables us to sample missing observations using the information in the available data.

For gamma GLMs, the transformation to normality uses the Anscombe residuals (see Appendix D). To generate a vector of dependent random variables in the GLM, we can draw a vector of correlated Anscombe residuals from the appropriate multivariate normal distribution and invert the residual transformation (which will depend on the means under the model). In general, Anscombe residual transformations are approximate rather than exact, but the approximation is extremely good in many cases.

Providing each pair of sites has an overlapping period of record, the residual correlation for that pair can be estimated straightforwardly. Unfortunately however, the resulting set of correlations for all pairs of sites is not guaranteed to be internally consisten. Moreover, correlations cannot be estimated for pairs of sites whose periods of records do not overlap, or for ungauged locations. To get round this, one might choose to adopt a valid spatial model for the correlations. For small spatial scales, inter-site correlations tend to be so similar that they can be regarded as effectively constant: in this case, the constant correlation can be estimated as a weighted average of the available inter-site correlations with weights proportional to the numbers of contributing observations for each pair of sites.

At larger spatial scales, however, it becomes necessary to account for the likely decay of correlation with inter-site distance (and potentially direction). There are several commonly-used spatial correlation models that can capture this behaviour. Currently the software offers a choice of exponential and powered exponential correlation models (see Table 7); also the option for the correlation to decay to a non-zero threshold rather than to zero at large distances. This phenomenon has been observed in several data sets, and probably arises from the relatively small scale of many study regions relative to the synoptic scales of the weather systems affecting them: the overall characteristics of an individual system are likely to affect all sites simultaneously on any given day, with enhanced inter-site dependence at

local scales within the system.

The correlation models currently handled by the software can all be written in the general form

$$\rho\left(d;\boldsymbol{\theta},\alpha\right)=\alpha+\left(1-\alpha\right)\rho^{*}\left(d;\boldsymbol{\theta}\right)\ , \tag{35}$$

where d is the inter-site distance, $\boldsymbol{\theta}$ is a parameter vector, α is the limiting correlation at large distances and $\rho^{*}\left(\cdot;\cdot\right)$ is a 'standard' correlation model that decays to zero at large distances. The parameters $(\boldsymbol{\theta},\alpha)$ are estimated by minimising a weighted least-squares objective function:

$$S\left(\boldsymbol{\theta},\alpha\right)=\sum_{i}n_{i}\left[r_{i}-\rho\left(d_{i};\boldsymbol{\theta},\alpha\right)\right]^{2}\ , \tag{36}$$

where the sum is over all pairs of sites with overlapping records; r_i is the residual correlation computed for the ith pair; n_i is the number of residuals contributing to this correlation; and d_i is the inter-site distance for this site pair.

Minimising **(36)** with respect to $\boldsymbol{\theta}$ and α can be achieved by equating the gradient vector to zero. Writing $\rho_i=\rho\left(d_i;\boldsymbol{\theta},\alpha\right)$ and $\rho_i^{*}=\rho^{*}\left(d_i;\boldsymbol{\theta},\alpha\right)$, we have

$$\frac{\partial S}{\partial\boldsymbol{\theta}}=-2(1-\alpha)\sum_{i}n_{i}\left(r_{i}-\rho_{i}\right)\frac{\partial\rho_{i}^{*}}{\partial\boldsymbol{\theta}} \tag{37}$$

$$\text{and}\qquad\frac{\partial S}{\partial\alpha}=-2\sum_{i}n_{i}\left(r_{i}-\rho_{i}\right)\left(1-\rho_{i}^{*}\right)\ . \tag{38}$$

For a given value of $\boldsymbol{\theta}$, **(38)** can be solved analytically to yield the corresponding optimal value of α as

$$\hat{\alpha}\left(\boldsymbol{\theta}\right)=\left[\sum_{i}n_{i}\left(r_{i}-\rho_{i}^{*}\right)\left(1-\rho_{i}^{*}\right)\right]\Big/\left[\sum_{i}n_{i}\left(1-\rho_{i}^{*}\right)^{2}\right]\ . \tag{39}$$

For most correlation models, however, **(37)** does not have an analytical solution. For a given value of α, the solution can in principle be found iteratively by specifying an initial estimate $\boldsymbol{\theta}^{(0)}$ and then iterating to convergence the Newton-Raphson scheme

$$\boldsymbol{\theta}^{(t+1)}=\boldsymbol{\theta}^{(t)}-\left[\left.\frac{\partial^{2}S}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}'}\right|_{\boldsymbol{\theta}^{(t-1)}}\right]^{-1}\left.\frac{\partial S}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}^{(t-1)}}\ . \tag{40}$$

The Hessian $\partial^2 S/\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}'$ can be derived from **(37)** as

$$\frac{\partial^{2}S}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}'}=-2\left(1-\alpha\right)\sum_{i}n_{i}\left[\left(r_{i}-\rho_{i}\right)\frac{\partial^{2}\rho_{i}^{*}}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}'}-\left(1-\alpha\right)\left(\frac{\partial\rho_{i}^{*}}{\partial\boldsymbol{\theta}}\right)\left(\frac{\partial\rho_{i}^{*}}{\partial\boldsymbol{\theta}}\right)'\right]\ .$$

The software implements a slightly modified version of this scheme in which, at each iteration, the value of α is also updated using **(39)**. A further refinement is that the adjustments in each iteration of **(40)** are scaled if necessary to ensure that the values of $\boldsymbol{\theta}$ always satisfy any necessary constraints for the underlying correlation model. The resulting algorithm is probably suboptimal, but it seems to be fairly stable and to produce reasonable results.

E.3 Spatial dependence — binary variables

Incorporating spatial dependence into binary sequences is much more difficult than for continuous variables. Several options are available within this software. They are documented fairly completely here since not all details have been published elsewhere.

Throughout this section we denote by S_t the number of sites we are studying on day t, and by $\mathbf{Y}_t = (Y_{1t} \ \ldots \ Y_{S_t t})'$ a vector of binary variables corresponding to rainfall occurrence at those sites. A logistic regression model allows us to calculate $E(Y_{st}) = p_{st}$, say.

E.3.1 Latent Gaussian variables

A conceptually simple approach to generating correlated binary variables is to start by generating a set of correlated Gaussian variables $\mathbf{Z} = (Z_1, \ldots, Z_{S_t})$ and then to define

$$Y_{st} = \begin{cases} 1 & \text{if } Z_s > \tau_{st} \\ 0 & \text{otherwise,} \end{cases} \tag{41}$$

where the thresholds $\tau_{st}, \ldots, \tau_{S_t t}$ are chosen to ensure that $P(Y_{st} = 1) = p_{st}$ as required by the logistic regression model. Specifically, if the $\{Z_s\}$ are all standard normal variables (i.e. with zero mean and unit variance), we need to set $\tau_{st} = -\Phi^{-1}(p_{st})$ where $\Phi(\cdot)$ is the distribution function of the standard normal distribution: standard algorithms are available for computing this.

In this setup, dependence between sites s_1 and s_2 can be induced by specifying a correlation, $\rho_{s_1 s_2}$ say, between Z_{s_1} and Z_{s_2}. As this correlation varies between -1 and $+1$, so the strength of dependence between $Y_{s_1 t}$ and $Y_{s_2 t}$ varies over its entire range. The idea seems first to have been used by Pearson (1901). In the present context, the 'latent correlations' between each pair of sites are chosen so as to match the proportion of days for which both sites experience rain. Specifically, suppose there are n days for which observations are available at both sites s_1 and s_2. Then the observed proportion of days for which both sites experience rain is $n^{-1} \sum_t Y_{s_1 t} Y_{s_2 t}$ and the expected proportion is $n^{-1} \sum_t P(Z_{s_1} > \tau_{s_1 t}, Z_{s_1} > \tau_{s_2 t})$ (the sums here are over days for which both sites have data). Thus we choose the correlation $\rho_{s_1 s_2}$ to solve the equation

$$\sum_t P(Z_{s_1} > \tau_{s_1 t}, Z_{s_1} > \tau_{s_2 t}) = \sum_t Y_{s_1 t} Y_{s_2 t} \ . \tag{42}$$

For a given value of $\rho_{s_1 s_2}$, the probability on the left-hand side of **(42)** can be calculated using algorithms for the bivariate normal distribution: the software uses that of Donnelly (1973). A numerical search is then carried out to solve the equation for $\rho_{s_1 s_2}$.

The approach as just outlined has two drawbacks. First, by comparison with the other binary dependence models discussed below it is relatively slow: latent correlations have to be estimated separately for each pair of sites, and each of these requires repeated evaluation of bivariate normal probabilities (the calculation of which is nontrivial despite the use of a

fast algorithm) in the search for a root of **(42)**. For a moderately large data set (say 40 or 50 years of daily data at 40 sites), it might take between 5 and 10 minutes to estimate these correlations on a relatively fast modern machine — obviously the computing time increases quadratically with the number of sites.

A second drawback, which is arguably more serious, is that the matrix obtained by solving **(42)** for each pair of sites is not guaranteed to be positive definite, and is therefore not necessarily a valid correlation matrix. This can cause problems in simulation, since simulation algorithms for the multivariate normal distribution (required to generate a realisation of $\mathbf{Z}$ on each day of simulation here) require that the correlation matrix is positive definite. In practice, to overcome this problem it is necessary to postprocess the individual correlations by fitting an appropriate spatial correlation model and then using the *modelled* correlations as inputs to the simulation program. The software uses the algorithm described in Appendix E.2 to fit spatial correlation models here.

A final problem, in some way related to the second drawback above, is that a solution to **(42)** is not guaranteed. This is because as $\rho_{s_1 s_2}$ varies from its minimum value of -1 to its maximum value of $+1$, it can be shown that $P(Y_{s_1 t} = Y_{s_2 t} = 1)$ varies correspondingly from $\max(p_{s_1 t} + p_{s_2 t} - 1, 0)$ to $\min(p_{s_1 t}, p_{s_2 t})$. Thus the right-hand side of **(42)** is constrained to lie in the range

$$\sum_t \left[\max(p_{s_1 t} + p_{s_2 t} - 1, 0), \min(p_{s_1 t}, p_{s_2 t})\right] .$$

However, the left-hand side is not so constrained and — either due to sampling variation or to slight mis-specification of the modelled marginal probabilities by the GLM — can occasionally produce values outside this range. If this occurs for a pair of sites, the software estimates the corresponding latent correlation as -1 or $+1$ as appropriate, and issues a warning message.

Given a positive definite correlation matrix, simulation of a dependent vector of rainfall occurrence indicators is straightforward using this dependence structure: for each day of simulation, calculate the marginal probabilities as predicted by the GLM, simulate a vector $\mathbf{Z}$ from a multivariate normal distribution with the specified correlation structure, and then threshold the simulated Zs according to equation **(41)**.

Imputation is more difficult, however. Essentially, what is required is to simulate from the conditional distribution of $\mathbf{Z}$ given the observed elements of $\mathbf{Y}_t$ and then to generate the missing elements by thresholding the generated Z_s as before. The problem is in simulating from the conditional distribution of $\mathbf{Z}$. A naïve algorithm is as follows:

1. Sample a value of $\mathbf{Z}$ from its unconditional distribution
2. If all of the observed elements of $\mathbf{Y}_t$ are consistent with the corresponding elements of $\mathbf{Z}$, continue; otherwise reject the sampled $\mathbf{Z}$ and return to step 1.

The problem with this algorithm is that if observations are available at many sites, the probability of simultaneously generating a $\mathbf{Z}$ that is consistent with *all* of the available

observations will typically be rather small. Therefore, many attempts will be required before an acceptable $\mathbf{Z}$ is generated in step 1; this makes the algorithm very slow.

As an alternative therefore, one might consider speeding up the algorithm by retaining those elements of $\mathbf{Z}$ that are consistent with the corresponding observations and then re-sampling the remainder from their distribution conditional on the retained elements; this procedure can then be iterated until all of the generated values are consistent with the observations. Unfortunately, it can be shown that this procedure is incorrect and does *not* lead to a sample from the distribution of $\mathbf{Z}$ given $\mathbf{Y}_t$.

The solution adopted in the software is to deal with the problem in two parts. Specifically, denote by $\tilde{\mathbf{Y}}_t$ the vector of *observed* components of $\mathbf{Y}_t$ and by $\tilde{\mathbf{Z}}$ the corresponding elements of $\mathbf{Z}$. Then the first step is to sample from the joint distribution of $\tilde{\mathbf{Z}}$ conditional on $\tilde{\mathbf{Y}}_t$; the second step is to sample the remaining elements of $\mathbf{Z}$ from their distribution given $\tilde{\mathbf{Z}}$ (which is straightforward since the joint distribution is multivariate normal) and to threshold these remaining sampled elements to impute the missing values of $\mathbf{Y}_t$. Efficient sampling from the distribution of $\tilde{\mathbf{Z}}|\tilde{\mathbf{Y}}_t$ itself is nontrivial: the algorithm implemented here is a Gibbs sampler in which the elements of $\tilde{\mathbf{Z}}$ are initialised by sampling each one independently from a normal distribution truncated at the corresponding threshold τ; and then repeatedly visiting each element of $\tilde{\mathbf{Z}}$ in turn and sampling from its distribution conditional on the current configuration of the remaining elements and of $\tilde{\mathbf{Y}}$. Specifically, according to the correlation model outlined above, the unconditional distribution of $\tilde{\mathbf{Z}}$ is multivariate normal with mean $\mathbf{0}$ and covariance matrix $\mathbf{\Sigma}$, say. Let $\tilde{Z}_i$ denote the ith element of $\tilde{\mathbf{Z}}$ and let $\tilde{Z}_{(-i)}$ denote the vector of the remaining elements. Moreover, let $\sigma^{(ii)}$ denote the ith diagonal element of $\mathbf{\Sigma}^{-1}$ and let $\mathbf{\Sigma}^{-1}_{(i,-i)}$ denote the ith row of $\mathbf{\Sigma}^{-1}$ with the ith element removed. Then, using standard results for conditioning in the multivariate normal distribution, as well as the formula for the inverse of a partitioned matrix **REFS!!!** , it may be shown that the distribution of $\tilde{Z}_i$ given $\tilde{\mathbf{Z}}_{(-i)}$ is normal with mean $-\mathbf{\Sigma}^{-1}_{(i,-i)}\tilde{\mathbf{Z}}_{(-i)}/\sigma^{(ii)}$ and variance $1/\sigma^{(ii)}$. To update the value of $\tilde{Z}_i$ therefore requires sampling from this normal distribution, truncated as appropriate for consistency with the corresponding observation $\tilde{Y}_i$ say.

In the present context, a single iteration of the Gibbs sampler consists of a sequence of updates in which every element of $\tilde{\mathbf{Z}}$ is visited once. If the procedure is repeated for a large number of iterations, the resulting $\tilde{\mathbf{Z}}$ will be sampled approximately from the required distribution. Obviously, there is a tradeoff here between accuracy and computation time: the higher the number of iterations, the closer will be the distribution of the sampled $\tilde{\mathbf{Z}}$ to the required target but the overall simulation time will increase. For the purposes of imputing binary wet/dry indicators however, excessive accuracy is probably unnecessary and speed of execution is a priority. The software therefore uses just 10 Gibbs iterations when imputing rainfall occurrence indicators using this spatial dependence model. This choice was made on the basis of plots such as Figures 6 and 7. These have been produced for a hypothetical network of five sites of which site 2 is known to be dry and the remainder are wet, and with latent inter-site correlations ranging from 0.61 to 0.96. The top panel of Figure 6 shows a random sample of 50 initial configurations for the Gibbs sampler in this situation, with each

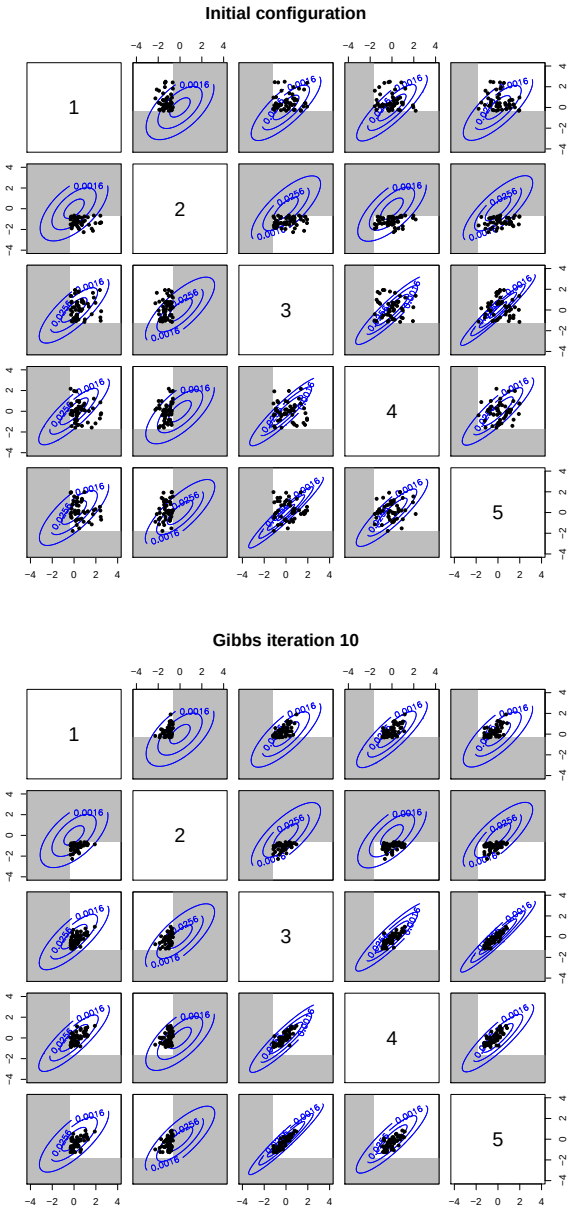

Figure 6: Gibbs sampling to draw from distribution of $\tilde{\mathbf{Z}}|\tilde{\mathbf{Y}}_t$ at five sites, with intersite correlations in $\tilde{\mathbf{Z}}$ ranging from 0.61 to 0.96. Site 2 is known to be dry and the remaining sites are wet; the grey region in each plot shows the values of $\tilde{\mathbf{Z}}$ that are inconsistent with these observations. Blue lines in each plot are contours of the bivariate densities from which the elements of $\tilde{\mathbf{Z}}$ would be drawn in the absence of observations. 50 separate realisations have been produced: each has been initialised by sampling the elements of $\tilde{\mathbf{Z}}$ independently from their marginal distributions (top plot). Bottom plot shows the sampled configurations after 10 iterations of the Gibbs sampler.

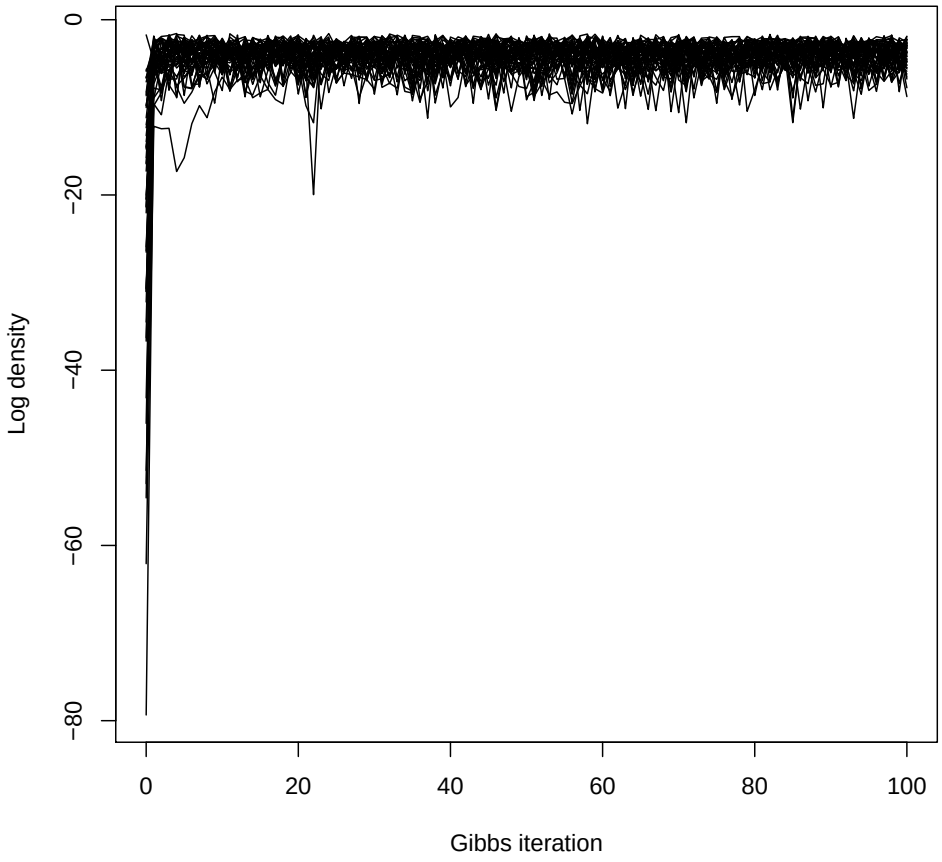

Figure 7: Convergence of the Gibbs sampler for $\tilde{\mathbf{Z}}$: logarithm of joint density of sampled observations over 100 Gibbs iterations, for each of 50 samples.

element of $\tilde{\mathbf{Z}}$ sampled independently as described above. The independent sampling shows up clearly: the sampled points are not aligned along the contours of the correlated joint distributions of the $\tilde{\mathbf{Z}}$-pairs. However, by the tenth iteration of the sampler (bottom panel) the points look much more plausibly like samples from these joint distributions. Figure 7 is another way of exploring the convergence of the sampler: here, at each iteration, the quantity $\log \pi\left(\tilde{\mathbf{Z}}\right)$ is plotted for every one of the 50 initial configurations, where $\pi(\cdot)$ is the joint density of a multivariate normal distribution with mean $\mathbf{0}$ and covariance matix $\mathbf{\Sigma}$. The figure shows that the initial configurations typically have a very low joint density, but that this rapidly increases within a few iterations to fluctuate around an equilibrium level indicating convergence to the required distribution. According to this plot, all but one of the 50 traces have reached the equilbrium level by around the tenth iteration.

E.3.2 Hidden binary weather state

Over a wide range of spatial scales, spatial dependence in binary climate sequences at a daily timescale (e.g. absence/occurrence of rainfall) is mainly due to the fact that all sites tend to be influenced by the same weather systems on particular days. This process can be modelled by including a hidden weather state which categorizes each day as 'on' or 'off' over the entire area. For incorporation into a logistic regression model, it is convenient to model the effect of such a hidden weather state on the log odds scale.

In this context, we consider that there is a random variable X_t associated with day t that represents the 'weather state' on that day. $X_t = 1$ with probability α_t (representing a 'wet day') and 0 otherwise (a 'dry day'). X_t is not directly observable.

The $\{Y_{st}\}$ are modelled as conditionally independent given X_t. The probability of rain at each site on day t is modelled using

$$\ln\left(\frac{P\left(Y_{st}=1|X_t=x\right)}{1-P\left(Y_{st}=1|X_t=x\right)}\right)=\ln\left(\frac{p_{st}}{1-p_{st}}\right)+x\ln a+\ln b_{st}\left(\alpha_t,p_{st},a\right)\ . \tag{43}$$

Here $\ln a$ is constant for all sites and days, and is free over the range $(-\infty,\infty)$. $\ln b_{st}(.)$ is a function of α_{st}, p_{st} and a, chosen to ensure that the unconditional probability of rain at site s is p_{st}. We abbreviate it to b_{st} for convenience. Some straightforward manipulation enables us to calculate b_{st} from the remaining parameters.

It can be shown that no matter how large a is, we cannot make $P(Y_{st}=1|X_t=1)$ arbitrarily close to 1 if $p_{st}<\alpha_t$. Similarly, we cannot make $P(Y_{st}=1|X_t=0)$ arbitrarily close to zero if $p_{st}>\alpha_t$. This indicates a limitation of this particular structure.

The covariance structure of model **(43)** can be derived. The covariance between Y_{s_1t} and Y_{s_2t} $(s_1\neq s_2)$ is defined as $P(Y_{s_1t}=Y_{s_2t}=1)-p_{s_1t}p_{s_2t}$, and is equal to

$$\operatorname{cov}\left(Y_{s_1t},Y_{s_2t}\right)=\frac{(1-\alpha)p_{s_1t}p_{s_2t}\left(1-p_{s_1t}\right)\left(1-p_{s_2t}\right)\left(1-b_{s_1t}\right)\left(1-b_{s_2t}\right)}{\alpha_t\left[1-p_{s_1t}\left(1-b_{s_2t}\right)\right]\left[1-p_{s_1t}\left(1-b_{s_2t}\right)\right]}\ . \tag{44}$$

Note that this covariance is not explicitly dependent on inter-site distance: thus this particular dependence structure is probably unsuitable for use in catchments that are large relative to the typical scale of weather systems that affect them so that inter-site correlations vary appreciably in magnitude for different pairs of sites.

This spatial dependence model involves two unknown parameters on any given day: α_t and a. An obvious choice for α_t is the mean of the $\{p_{st}\}$ on day t. In this case the marginal predictions of the GLM at all sites are used to determine whether a day will be 'wet' or 'dry'. Thus features such as seasonality are automatically incorporated into the weather states, since these should be reflected in the $\{p_{st}\}$.

How to choose a is less obvious because the sequence of Xs is not observed. Note, however, that given the ps predicted by the GLM, the corresponding α can be calculated. In this case, given the observed Y values it is possible, numerically, to obtain a maximum likelihood estimate of a. However, this may be undesirable since the model structure, while plausible

in some applications, is probably a fairly crude approximation of reality. An alternative estimation procedure is a method of moments. In applications, it is often important to reproduce the spatial coverage of a binary random field (i.e. the proportion of 1s among the sites). The software therefore chooses a so as equate the observed and theoretical variances of the coverage distribution. The theoretical variance can be derived from the expression for covariances at **(44)**. The observed variance can be calculated straightforwardly from the time series of coverages, weighting each day by the number of active sites. Equating the two can be achieved using straightforward numerical methods.

Imputation of missing data is straightforward under this spatial dependence structure. Suppose on a particular day we observe $Y_{1t} = y_1, \ldots, Y_{kt} = y_k$ $(s < S_t)$, and wish to fill in the remaining values $Y_{(k+1)t}, \ldots, Y_{S_t t}$. Rearranging **(43)**, we find that

$$P(Y_{1t} = y_1, \ldots, Y_{kt} = y_k | X_t = x) = \prod_{y_s=1} \frac{a^x b_{st} p_{st}}{1 - p_{st}\left(1 - a^x b_{st}\right)} \prod_{y_s=0} \left(1 - \frac{a^x b_{st} p_{st}}{1 - p_{st}\left(1 - a^x b_{st}\right)}\right) .$$

Now we can use Bayes' Theorem to find the conditional probability distribution of X_t on the basis of the observed Ys:

$$P(X_t = 1 | Y_{1t} = y_1, \ldots, Y_{kt} = y_k) = \frac{P(Y_{1t} = y_1, \ldots, Y_{kt} = y_k | X_t = 1)}{\sum_{x=0}^{1} P(Y_{1t} = y_1, \ldots, Y_{kt} = y_k | X_t = x) P(X_t = x)} . \tag{45}$$

Recalling that $P(X_t = 1) = \alpha = 1 - P(X_t = 0)$, we can therefore impute missing data by simulating $X_t = 1$ with probability given by **(45)**, and then sampling the remaining missing sites independently from **(43)** as before.

E.3.3 Modelling the coverage distribution

One feature of the hidden weather state model above is that the probability of all sites being in the same state (0 or 1) cannot be made arbitrarily close to 1. This may not be a problem for some applications; however, in some problems (for example rainfall modelling at small spatial scales) it is common for the distribution of coverage to be 'U-shaped' with a high concentration at both 0 and 1. An alternative approach to the generation of dependent binary sequences is to address the dependence directly through the coverage distribution, thus guaranteeing that simulations will reproduce this important feature.

In this alternative model therefore, a (non-unique) dependence structure can be specified for $\mathbf{Y}_t$ through the distribution of $Z_t = \sum_{s=1}^{S_t} Y_{st}$. Since the marginal occurrence probabilities (i.e. the p_{st}s) vary from day to day, so does the distribution of Z_t — in particular, we have $\mathrm{E}(Z_t) = \sum_{s=1}^{S_t} p_{st}$.

A flexible family of distributions for discrete random variables taking values in $\{0, 1, \ldots, S_t\}$ is the Beta-Binomial family:

$$P(Z_t = z) = \binom{S_t}{z} \frac{\Gamma(\alpha_t + z)\,\Gamma(S_t + \beta_t - z)\,\Gamma(\alpha_t + \beta_t)}{\Gamma(\alpha_t + \beta_t + S_t)\,\Gamma(\alpha_t)\,\Gamma(\beta_t)} \qquad (z = 0, 1, \ldots, S_t) , \tag{46}$$

for some parameters $\alpha_t, \beta_t \in \mathbb{R}^+$. For small values of S_t, these probabilities can be evaluated cheaply using a recurrence relation. The mean and variance of the distribution are

$$\frac{S_t \alpha_t}{\alpha_t + \beta_t} \qquad \text{and} \qquad \frac{S_t \alpha_t \beta_t \left(\alpha_t + \beta_t + S_t\right)}{\left(\alpha_t + \beta_t\right)^2 \left(\alpha_t + \beta_t + 1\right)} \ , \tag{47}$$

respectively. The uniform distribution corresponds to the special case $\alpha_t = \beta_t = 1$. If $\alpha_t = 0, \beta_t \neq 0$ then $P\left(Z_t = 0\right) = 1$, and if $\beta_t = 0, \alpha_t \neq 0$ then $P\left(Z_t = S_t\right) = 1$. The case when $\alpha_t = \beta_t = 0$ is discussed below.

It is convenient to reparametrise the Beta-Binomial distribution here: set

$$\theta_t = \frac{\alpha_t}{\alpha_t + \beta_t} \qquad \text{and} \qquad \phi_t = \alpha_t + \beta_t \ , \tag{48}$$

so that

$$\alpha_t = \theta_t \phi_t \ , \qquad \beta_t = \phi_t \left(1 - \theta_t\right) \ , \qquad \mathrm{E}\left(Z_t\right) = S_t \theta_t \text{ and } \mathrm{Var}\left(Z_t\right) = \frac{S_t \theta_t \left(1 - \theta_t\right) \left(\phi_t + S_t\right)}{\phi_t + 1} \ . \tag{49}$$

We can think of θ_t as a mean value parameter, and ϕ_t as a dispersion parameter (in fact, ϕ_t essentially controls the tendency of the distribution to be concentrated either at its extremities or around its mean). It is convenient, and not implausible, to assume that $\phi_t = \phi$ is constant for all t, so that θ_t (which is known, since it can be calculated from the logistic regression model) is the only time-varying parameter of the distribution. As θ_t varies then, so we hope to reproduce typical 'summer' and 'winter' coverage distributions for example.

The reparametrisation also allows us to investigate the shape of the distribution when $\alpha_t = \beta_t = 0$. In this case, we have $\phi = 0$. If we consider $\lim_{\phi \to 0} P\left(Z_t = 0\right)$ with θ_t fixed, we find that in the limit Z_t takes the values 0 and S_t with probabilities $1 - \theta_t$ and θ_t respectively. At the other extreme, it can be shown that the limiting distribution as $\phi \to \infty$ is Binomial with parameters S_t and θ_t. Since the Binomial arises if all the Y_{st}s are independent and identically distributed, we see that ϕ can be regarded as an overall summary of the dependence among the Y_{st}s — small values of ϕ correspond to strong dependence.

Given data $\{(S_t, Z_t, \theta_t) : t = 1, \ldots, T\}$, a natural way to estimate the dispersion parameter ϕ is via a method of moments. Note that

$$\mathrm{E}\left(\frac{\left(Z_t - S_t \theta_t\right)^2}{S_t \theta_t \left(1 - \theta_t\right)}\right) = \frac{\phi + S_t}{\phi + 1} = E\left(R_t^2\right) \quad \text{say.}$$

Hence

$$\mathrm{E}\left(\sum_{t=1}^{T} R_t^2\right) = T + \frac{1}{\phi + 1} \sum_{t=1}^{T} \left(S_t - 1\right) \ ,$$

so that a natural estimator of ϕ is

$$\hat{\phi} \ = \ \frac{\sum_{t=1}^{T} \left(S_t - 1\right)}{\sum_{t=1}^{T} \left(R_t^2 - 1\right)} - 1 \ . \tag{50}$$

In the simple case where all the p_{st}s are equal and the Ys are independent, we have $Z_t \sim Bin(S_t, \theta_t)$, and $\mathrm{Var}(Z_t) = S_t\theta_t(1-\theta_t)$, whence $\mathrm{E}(R_t^2) = 1$. If all the R_t^2s were equal to 1, **(50)** would give $\hat{\phi} = \infty$ (corresponding to independence): overdispersion results in lower values of $\hat{\phi}$ as expected.

To simulate from this model, a natural strategy is to sample the number of wet sites from the distribution of Z_t, and then to allocate the positions of these wet sites. However, this will only yield sequences with the correct properties if the conditional probabilities of rain at each site, given the overall number of wet sites, are correctly specified. This can be achieved providing a valid joint distribution can be found for $\mathbf{Y}_t$ and Z_t. We now describe the algorithm used to find such a joint distribution (which may not be unique). The subscript t is now unnecessary, so we drop it and write $\mathbf{Y}, Z$. We assume initially that the given distribution of Z is compatible with the individual probabilities of the Ys (this is not guaranteed, as we will see below).

From our earlier notation, we have $P(Y_s = 1) = p_s$. We also define $\pi_z = P(Z = z)$, and $w_{s,z} = P(Y_s = 1 \text{ and } Z = z)$. A first step in determining a joint distribution of $\mathbf{Y}$ and Z is to find $\{w_{s,z} : s = 1, \ldots, S; z = 0, \ldots, S\}$, from which we can calculate the conditional probabilities at each site:

$$P(Y_s = 1 | Z = z) = \frac{w_{s,z}}{\pi_z} \ . \tag{51}$$

The following relationships must hold:

$$0 \leq w_{s,z} \leq \pi_z \ ; \tag{52}$$

$$\sum_{z=0}^{S} w_{s,z} = p_s \ ; \quad \text{and} \tag{53}$$

$$\sum_{s=1}^{S} w_{s,z} = z\pi_z \ . \tag{54}$$

The second of these is the Law of Total Probability; the third can be seen by noting that $\mathrm{E}\left[\left(\sum_s Y_s\right) \middle| Z = z\right] = z$. But

$$\mathrm{E}\left[\left(\sum_s Y_s\right) \middle| Z = z\right] = \sum_s P(Y_s = 1 | Z = z) = \sum_s \frac{w_{s,z}}{\pi_z} \ ,$$

and the condition follows.

To visualise the problem, it is helpful to lay the ws out in the form of a contingency table, as in Figure 8. The only constraint which is not apparent from this is **(52)**.

The algorithm developed here is based on linear programming ideas, but takes advantage of the rather tight constraints to speed up the search for the ws. Additionally, it is insensitive to the order in which sites are considered . The basic procedure is to allocate a row of the table at a time, starting with $w_{s,0} = 0$ $(s = 1, \ldots, S)$ and noting that $w_{s,S} = \pi_S$ $(s = 1, \ldots, S)$

		s				
		1	2	$\cdots$	S	**TOTAL**
z	0	0	0	$\cdots$	0	0
	1	$w_{1,1}$	$w_{2,1}$	$\cdots$	$w_{S,1}$	π_1
	2	$w_{1,2}$	$w_{1,2}$	$\cdots$	$w_{S,2}$	$2\pi_2$
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
	S	π_S	π_S	$\cdots$	π_S	$S\pi_S$
TOTAL		p_1	p_2	$\cdots$	p_S	$\mathrm{E}(Z)$

Figure 8: Contingency table illustrating restrictions on the weights $\{w_{s,z} : s = 1, \ldots, S; z = 0, \ldots, S\}$.

— both of which follow from constraints (52) and (54) above. As each row is allocated, the constraints on the remaining entities will change. Specifically, suppose we have allocated rows $0, 1, \ldots, z-1$ and are currently considering row $z \leq S-1$. Constraints (52) and (54) above are unchanged. If we define $p_{s|z} = P(Y_s = 1 \text{ and } Z \geq z)$, then constraint (53) becomes

$$\sum_{j=z}^{S} w_{s,z} = p_{s|z} = p_s - \sum_{j=0}^{z-1} w_{s,z} \ . \tag{55}$$

Since we know that $w_{s,S} = \pi_S$, we must have $w_{s,z} \leq p_{s|z} - \pi_S$ for $z \leq S-1$. A further inequality can be deduced from constraint (52) applied to the subsequent rows of the table: $p_{s|z+1} = \sum_{j=z+1}^{S} w_{s,j} \leq \sum_{j=z+1}^{S} \pi_j \Rightarrow p_{s,z} - w_{s,z} \leq \sum_{j=z+1}^{S} \pi_j$, so that $w_{s,z} \geq p_{s|z} - \sum_{j=z+1}^{S} \pi_j$. Putting these inequalities together we must have, for $z \leq S-1$,

$$\max\left(0, p_{s|z} - \sum_{j=z+1}^{S} \pi_j\right) \leq w_{s,z} \leq \min\left(\pi_z, p_{s|z} - \pi_S\right) \ . \tag{56}$$

Writing the lower and upper bounds as $LB_{s,z}$ and $UB_{s,z}$ respectively, and summing, gives

$$\sum_{s=1}^{S} LB_{s,z} \leq \sum_{s=1}^{S} w_{s,z} \leq \sum_{s=1}^{S} UB_{s,z} \ . \tag{57}$$

Now from (54) above, we require $\sum_{s=1}^{S} w_{s,z} = z\pi_z$. Hence, providing $\sum_{s=1}^{S} LB_{s,z} \leq z\pi_z \leq$

$\sum_{s=1}^{S} UB_{s,z}$, we can set

$$w_{s,z} = LB_{s,z} + \frac{z\pi_z - \sum_{s=1}^{S} LB_{s,z}}{\sum_{s=1}^{S} UB_{s,z} - \sum_{s=1}^{S} LB_{s,z}} \tag{58}$$

for each s, and proceed to the next row of the table. If the desired row total $z\pi_z$ is outside the interval $\left[\sum_{s=1}^{S} LB_{s,z}, \sum_{s=1}^{S} UB_{s,z}\right]$ then we must return to a previous row and re-allocate some of the joint probabilities. The following result is useful:

Result: providing the entries in rows 0 to $z-1$ of the table all satisfy inequalities of the form **(56)**, the inequality $\sum_{s=1}^{S} UB_{s,z} \geq z\pi_z$ is automatically satisfied.

The proof is omitted, since it is not particularly instructive and the margin is too small to contain it. ■

The result tells us that in our algorithm, the only problem that can arise is when $\sum_{s=1}^{S} LB_{s,z} > z\pi_z$. Since $LB_{s,z} = \max\left(0, p_{s|z} - \sum_{j=z+1}^{S} \pi_j\right)$, the only way around this is to re-allocate some probabilities so as to reduce $p_{s|z}$ at sites where $p_{s|z} - \sum_{j=z+1}^{S} \pi_j > 0$ (because the πs are fixed), with a corresponding increase at sites where $p_{s|z} - \sum_{j=z+1}^{S} \pi_j < 0$. If this cannot be achieved, the given distribution of Z is incompatible with the marginal probabilities of the Ys.

We are now in a position to summarise the algorithm for calculating the ws. For each z:

1. Compute $p_{s|z} = p_s - \sum_{j=1}^{z-1} w_{s,j}$.
2. Compute $LB_{s,z} = \max\left(0, p_{s|z} - \sum_{j=z+1}^{S} \pi_j\right)$ and $UB_{s,z} = \min\left(\pi_z, p_{s|z} - \pi_S\right)$.
3. Compute $\sum_{s=1}^{S} LB_{s,z}$ and $\sum_{s=1}^{S} UB_{s,z}$.
4. If $\sum_{s=1}^{S} LB_{s,z} \leq z\pi_z$, calculate the ws for the current row according to **(58)**. Otherwise, re-allocate some probabilities in previous rows of the table, if possible. For practical purposes, when re-allocating probabilities we try to avoid setting any value of $w_{s,z}$ to either 0 or π_z (except when $z = 0$ or S), since this can lead to problems in imputation (see below) and is unrealistic.

The joint distribution of $\mathbf{Y}$ and Z is only partially specified by the ws in Figure 8, since the dependencies among the Ys are not represented. However, for simulation purposes it is not necessary to specify the distribution completely: all that is required is to sample one site at a time and then update the probabilites at the remaining sites to condition upon the sampled value. It can be shown that this updating can be done using the algorithm of the previous section.

The discussion so far has assumed that the distribution of Z is compatible with the marginal probabilities of the Ys. This is not guaranteed — obvious examples of incompatibility arise when $P(Z = S) > \min_s P(Y_s = 1)$ and when $P(Z = 0) > \min_s P(Y_s = 0)$. When simulating long sequences, it is almost inevitable that at some stage we will encounter a situation where the specified probabilities are incompatible with the distribution of Z. If this occurs, to continue simulation we must either adjust the probabilities or the distribution of Z.

In practice, incompatibility usually occurs when one or two of the individual ps are very different from the majority — in many situations, this is unrealistic (it is usually related to dependence upon previous days' values, and the problem can be reduced by including averages of previous days' values as covariates). Since the objective of this spatial dependence model is to reproduce a plausible distribution for the number of wet sites, the software deals with incompatibility by modifying the ps rather than the distribution of Z. Specifically, we shrink the ps towards their mean i.e. for each s replace p_s with

$$p_s - \lambda (p_s - \theta) \ , \tag{59}$$

where $\lambda \in (0, 1)$, and θ is the mean of the ps (and the mean-value parameter of the beta-binomial distribution — see equation **(49)**). The expected number of wet sites is unchanged by this adjustment. For small values of λ, the adjustment is a small one, in which case it may need to be repeated until a compatible set of probabilities is found. Repeated adjustments are guaranteed to find a compatible set of probabilities, since in the limit all of the probabilities are equal and, in this limit, it can be shown that at least one joint distribution exists.

Imputation using this model is, again, straightforward. Without loss of generality, we assume that the values of $Y_1, \ldots, Y_k$ are observed, and that $Y_{k+1}, \ldots, Y_S$ are missing. The starting point is the table of joint probabilities $\{w_{s,z}\}$. We work with the 'observed' sites one at a time, at each stage updating both the distribution of Z, and the probabilities of rain at the remaining sites, to condition on the observations. The procedure differs from the 'unconditional' case only insofar as we take the observed values of $Y_1, \ldots, Y_k$ rather than simulating them.

Known bugs and problems

The following problems are known to occur within the software. Please provide further bug reports to me (`richard@stats.ucl.ac.uk`) in the unlikely event of there being any ...

- For Windows systems running both 32- and 64-bit R installations, problems have been encountered when running Rglimclim under 32-bit R (a common symptom is a long string of error messages about gamma or digamma functions when fitting models). I don't understand the cause of this, since the supplied Windows distribution contains both 32- and 64-bit binaries. The resolution is to use 64-bit R where possible, although I would welcome any suggestions for more satisfactory solutions.
- Sometimes the fitting and simulation programmes produce the following error message:

  ```
  Input file error: last successful access was for data on  0/ 0/   0.
  ```

  This is caused by the underlying Fortran code getting confused about which files are open, and is usually the result of a previous error. The solution is to type

  ```
  > CloseFiles(10:1000)
  ```

  at the R prompt. This will be fixed at some point ...
- If a definition file created under Dos or Windows is used on a Unix system, character strings may read incorrectly. The reason is the extra line feed character (`^M`) used by Dos/Windows: the software reads this as part of the input. This problem may manifest itself via site names or attributes appearing incorrectly (if at all) in output files. The fix is to run a utility such as `dos2unix` / `fromdos` / whatever-the-name-is-on-your-system, on the definition file first.
- Conversely, if a definition or data file created under Unix is used on a Windows system, the absence of line feed characters can cause problems and produce 'end of file' error messages. In Windows, the easiest way to fix this is to open the offending definition / data file in WordPad and save it without making any changes: this will automatically append the required line feed characters.
- If the fitting programs terminate abnormally, they can leave large temporary files lying around. These need to be manually deleted.

Revision history

16th October 2013: Version (1.0-1):

- Corrected an inconsistency in the code for reading external predictor values from file: the missing value flag was incorrectly set in the code as -9999.99 instead of -9999.9.
- Corrected an error in the internal code for pseudo-random number generation, to ensure that if the random number seed is reset within an R session then the Rglimclim internal random number generator is completely reset (the original version did not reset array pointers; nor did it prevent the use of 'spare' random normal deviates from the Box-Müller algorithm).
- Corrected the routine that checks for the existence of required input files when simulating, so that files of empirical correlations are no longer required to be present unless the model specifically needs them;
- Corrected an error in the code for plotting inter-site correlations;
- Corrected an error in the `anova` method for fitted model objects, relating to comparisons of models with different numbers of site effects;
- Made some changes to the manual (e.g. added Appendix A.2 on multivariate modelling).

8th October 2013: First public release (1.0-0) of Rglimclim, building on the final Fortran version of Glimclim.

References

Chandler, R. E. (2005). On the use of generalized linear models for interpreting climate variability. *Environmetrics*, 16, no. 7:699–715.

Chandler, R. E. and Bate, S. M. (2007). Inference for clustered data using the independence log-likelihood. *Biometrika*, 94:167–183.

Chandler, R. E. and Scott, E. M. (2011). *Statistical Methods for Trend Detection and Analysis in the Environmental Sciences,*. Wiley, Chichester.

Chandler, R. E. and Wheater, H. S. (2002). Analysis of rainfall variability using Generalized Linear Models — a case study from the West of Ireland. *Water Resources Research*, 38, No.10:doi:10.1029/2001WR000906.

Cox, D. R. and Hinkley, D. (1974). *Theoretical Statistics*. Chapman & Hall, London.

Davison, A. C. (2003). *Statistical Models*. Cambridge University Press, Cambridge.

Dawid, A. P. (1986). Probability forecasting. In Kotz, S. and Johnson, N., editors, *Encyclopedia of Statistical Sciences*, pages 210–218. Wiley, New York.

Dobson, A. J. (2001). *An Introduction to Generalized Linear Models (second edition)*. Chapman and Hall, London.

Donnelly, T. G. (1973). Algorithm 462: bivariate normal distribution. *Communications of the ACM*, 16(10):638.

Fahrmeir, L. and Tutz, G. (1994). *Multivariate Statistical Modelling Based on Generalized Linear Models*. Springer-Verlag, New York.

Jørgensen, B. (1983). Maximum likelihood estimation and large-sample inference for generalized linear and nonlinear regression models. *Biometrika*, 70:19–28.

Krzanowski, W. J. (1998). *An introduction to statistical modelling*. Arnold, London.

Liang, K.-Y. and Zeger, S. (1986). Longitudinal data analysis using generalized linear models. *Biometrika*, 73, no.1:13–22.

Marsaglia, G. and Zaman, A. (1991). A new class of random number generators. *Ann. Appl. Prob.*, 1:462–480.

McCullagh, P. and Nelder, J. (1989). *Generalized Linear Models (second edition)*. Chapman and Hall, London.

Milne, A. A. (1958). *The World of Pooh (the complete Winnie-the-Pooh and The House at Pooh Corner)*. Methuen, London.

Pearson, K. (1901). Mathematical contributions to the theory of evolution — VII. On the correlation of characters not quantitatively measurable. *Phil. Trans. Roy. Soc. Series A*, 195:1–47.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1992). *Numerical Recipes in FORTRAN (second edition)*. Cambridge University Press.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Wei, B.-C. (1997). *Exponential Family Nonlinear Models*, volume 130 of *Lecture Notes in Statistics*. Springer, Singapore.

Wheater, H. S., Isham, V. S., Onof, C., Chandler, R. E., Northrop, P. J., Guiblin, P., Bate, S. M., Cox, D. R., and Koutsoyiannis, D. (2000). Generation of spatially consistent rainfall data. Report to the Ministry of Agriculture, Fisheries and Food (2 volumes). Also available as Research Report no. 204, Department of Statistical Science, University College London (`http://www.ucl.ac.uk/Stats/research/Resrpts/abstracts.html`).

Yan, Z., Bate, S., Chandler, R. E., Isham, V. S., and Wheater, H. S. (2002). An analysis of daily maximum windspeed in northwestern Europe using Generalized Linear Models. *J. Climate*, 15, No.15:2073–2088.

Yan, Z., Bate, S., Chandler, R. E., Isham, V. S., and Wheater, H. S. (2005). Changes in extreme wind speeds in NW Europe simulated by Generalized Linear Models. *Theoretical and Applied Climatology*. In press.

Yang, C., Chandler, R. E., Isham, V. S., Annoni, C., and Wheater, H. S. (2005). Simulation and downscaling models for potential evaporation. *J. Hydrol.*, 302:239–254.