MOMFIT

Software for moment-based fitting of single-site stochastic rainfall models

User Guide

Richard Chandler, Georgios Lourmas and Joao Jesus
Department of Statistical Science, University College London
September 29, 2016

Contents

Licence Agreement

The copyright in this user manual and the accompanying software, distributed in file `momfit.zip` (together 'the Software') is owned by University College London of Gower Street, London, WC1E 6BT ('UCL'). By proceeding to use the Software you (an individual or any other legal entity) agree to be bound by the terms of this Agreement which will govern your use of the Software.

1. **Licence**

   1.1 You are permitted:

   (a) to load the Software into and use it on a single computer which is under your control;

   (b) to transfer the Software from one computer to another provided it is used on only one computer at any one time; and

   (c) to transfer the Software (complete with all its associated documentation) and the benefit of this Agreement to another person provided they have agreed to accept the terms of this Agreement and you contemporaneously transfer all copies of the Software you have made to that person or destroy all copies not transferred. If any transferee does not accept such terms then this Agreement shall automatically terminate. The transferor does not retain any rights under this Agreement in respect of the transferred Software.

   (d) to use the software for academic use only. By academic use it is meant that you can only use the software within an recognised academic institution and then only for the purposes of research and study. Any use of the software not in accordance with the previous sentence and also if used, whether directly or indirectly, for any commercial activity shall automatically terminate this Licence.

   1.2 You are not permitted:

   (a) to use or copy the Software other than as permitted by this Licence;

   (b) to load the Software on to a network server for the purposes of distribution to one or more other computer(s) on that network or to effect such distribution (such use requiring a separate licence);

   (c) except as expressly permitted by this Agreement and save to the extent and in the circumstances expressly required to be permitted by law, to rent, lease, sub-license, loan, copy, modify, adapt, merge, translate, reverse engineer, decompile, disassemble or create derivative works based on the whole or any part of the Software or its associated documentation or use, reproduce or deal in the Software or any part thereof in any way.

2. **Duration**

   This Agreement is effective until you terminate it by destroying the Software and its documentation together with all copies. It will also terminate if you fail to abide by its terms. Upon termination you agree to destroy all copies of the Software and its documentation including any Software stored on the hard disk of any computer or floppy disk or other removable media under your control.

3. **Exclusion of Warranties**

   3.1 You accept and acknowledge that this License does not set out any warranty in respect of the Software other than that save as expressly provided for in this Agreement and any condition or warranty implied by law as to the quality or fitness for purpose of the Software or as to any services provided hereunder in relation to the Software is hereby excluded to the fullest extent permitted by law. For the avoidance of doubt, UCL gives no warranty, in respect of:

   (a) Any failure of the Software to operate due to changes in the operating environment or in any operating system; or

   (b) Any failure of the functions provided by the Software to meet your requirements or to operate in combination with any hardware or other software which you may select for its use.

   3.2 You acknowledge and accept:

   (a) That the Software is still under development and will be for test and evaluation purposes only

   (b) That UCL has not produced the Software to meet your own specification;

   (c) That the Software cannot be tested in every possible combination and operating environment and that it is not possible to produce economically (if at all) computer programs known to be error free or which operate in an uninterrupted manner and that not all errors are necessarily capable of rectification.

   3.3 UCL shall not be liable to you for any indirect or consequential loss, damage or expense of any kind whatsoever arising out of or in connection with the Software whether arising in contract, tort, negligence, breach of statutory duty or otherwise.

   3.4 Subject always to clause 3.3, UCL's liability in contract, tort, negligence, breach of statutory duty or otherwise with respect to any claim arising in respect of its acts or omissions under or in connection with this Agreement shall be limited to the sums received by UCL at the date of the claim relating to such act or omission or UK £1,000,000 whichever is the lesser.

4. **Intellectual Property**

   All copyright, trade marks and other intellectual property rights subsisting in or used in connection with the Software (including but not limited to all images, animations, audio and other identifiable material relating to the Software) are and remain the sole property of UCL.

5. **Law**

   This Agreement shall be governed by English law.

Acknowledgements

The initial development and testing of this software was largely funded by DEFRA R&D project FD2105 *Generation of Spatially Consistent Rainfall Data*. Georgios Lourmas' contribution was funded through the DYNSTOCH Network, a research training network running from 2000–2004 under the programme *Improving Human Potential* financed by the The Fifth Framework Programme of the European Commission (see `http://www.math.ku.dk/~michael/dynstoch/`). Joao Jesus' contribution was funded through an Engineering and Physical Sciences Research Council Doctoral Training Grant, 2007–2010.

The work leading to this software has benefited throughout from helpful discussions with numerous colleagues, notably Valerie Isham, Paul Northrop, Christian Onof and Howard Wheater.

Update history

September 2016: fixed a bug in the example and `fit_demo.r` script that caused errors in later versions of R. Thanks to Esmaeel Dodangeh for flagging this.

June 2010:

- Added skewness as a fitting property for the basic Poisson model.
- Corrected an error in the expression for `Pdry` in basic Neyman-Scott model (`NSO`).
- Extended the software to allow estimation via minimisation of a quadratic form rather than just a weighted sum of squares.
- Added option to calculate objective function derivatives using an analytic approximation rather than numerically, which should robustify the calculation of standard errors slightly.

January 2006: Bug fixes, some redefinitions (notably replacing the proportion of dry intervals with the proportion of wet intervals as a fitting property — hence the assignation `Ph=1-PH` in most routines).

2000–2004: Initial routines developed.

1 Purpose

This software is designed to fit stochastic single-site rainfall models to data using a method of moments. Specifically, given:

1. a vector $\mathbf{T} = (T_1 \ \ldots \ T_k)'$ of observed rainfall summary statistics (means, variances, autocorrelations etc.); and
2. a rainfall model, whose properties depend on a parameter vector $\boldsymbol{\theta}$,

let $\boldsymbol{\tau}(\boldsymbol{\theta}) = (\tau_1(\boldsymbol{\theta}) \ \ldots \ \tau_k(\boldsymbol{\theta}))'$ be the expected value of $\mathbf{T}$ according to the model. The idea behind the method of moments is to choose $\boldsymbol{\theta}$ to minimise a quadratic form

$$S(\boldsymbol{\theta}) = [\mathbf{T} - \boldsymbol{\tau}(\boldsymbol{\theta})]' \mathbf{W} [\mathbf{T} - \boldsymbol{\tau}(\boldsymbol{\theta})] \tag{1}$$

where $\mathbf{W}$ is a $k \times k$ matrix of 'weights'. Historically, in the rainfall modelling literature the kinds of models considered here have been fitted by minimising a weighted sum of squares between observed and expected properties:

$$S(\boldsymbol{\theta}) = \sum_{i=1}^{k} w_i [T_i - \tau_i(\boldsymbol{\theta})]^2 \ , \tag{2}$$

where $\{w_i : i = 1, \ldots, k\}$ is a collection of positive weights. This is a special case of **(1)**, in which the matrix $\mathbf{W}$ is diagonal with ith diagonal element w_i.

Denoting by $\hat{\boldsymbol{\theta}}$ the estimator obtained by minimising **(1)**, it may be shown that in large samples, the distribution of $\hat{\boldsymbol{\theta}}$ is approximately multivariate normal. The mean of this multivariate normal distribution is $\hat{\boldsymbol{\theta}}_0$ (the 'target' value, which is equal to the true value of the parameter if the data were indeed generated from the model being fitted). The covariance matrix is $\mathbf{H}^{-1}\mathbf{J}\mathbf{H}^{-1}$, where $\mathbf{J}$ is the covariance matrix of the gradient vector $\partial S/\partial \boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}_0$ and $\mathbf{H} = \mathrm{E}\left(\partial^2 S/\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}'|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}\right)$ is the expected Hessian. This result can be used to construct approximate standard errors and confidence intervals for the parameter estimates, providing $\mathbf{J}$ and $\mathbf{H}$ can be estimated. An estimator of $\mathbf{J}$ can be constructed from the covariance matrix of the vector $\mathbf{T}$ of summary statistics. The obvious way to estimate $\mathbf{H}$ is by evaluating the Hessian of **(1)** at the minimum: $\hat{\mathbf{H}} = \partial^2 S/\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}'|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}$. An alternative estimator, which can be shown to be asymptotically equivalent (although this is not at all obvious!), is

$$\tilde{\mathbf{H}} = 2\left[\partial\boldsymbol{\tau}/\partial\boldsymbol{\theta}|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}\right]' \mathbf{W} \left[\partial\boldsymbol{\tau}/\partial\boldsymbol{\theta}|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}\right] \ . \tag{3}$$

The latter estimator seems to be more numerically stable in practice.

Minimisation of **(1)** must usually be done numerically, which can be nontrivial. This software implements the minimisation and associated covariance matrix calculations for a variety of single-site models, in what is hopefully a flexible, reliable and user-friendly manner.

A rudimentary knowledge of the S language (as implemented in R and S-Plus, for example) is required — in particular, an understanding of data structures such as lists, vectors and data frames.

Further details on the models implemented in the software, including their parametrisation and properties, are given in DEFRA R&D project FD2105 report no. 8 *Mathematical expressions of generalised moments used in single-site rainfall models* by Christian Onof. The theory underlying the method of moments is summarised in FD2105 report no. 7 *Moment-based inference for stochastic-mechanistic models* by Richard Chandler. PDF versions of both reports are included in the software distribution. They will be referred to respectively as 'Report no. 8' and 'Report no. 7' below.[1]

2 Package contents and system requirements

The software is provided as a single R script, `momfit.r`. R version 1.7.0 or later is required to run the routines. R is freely available from `http://www.R-project.org/`. Installations are available for most operating systems, and most R code is completely portable between systems. We have used the software on Windows and Linux platforms. The routines run substantially faster (a factor of 2 or 3) under Linux, but otherwise we have not found any differences between platforms.

In addition to `momfit.r`, the package contains the following files:

`manual.pdf`: This manual.

`report7.pdf`: PDF copy of FD2105 report no. 7 *Moment-based inference for stochastic-mechanistic models* (see above).

`Report8.pdf`: PDF copy of FD2105 report no. 8 *Mathematical expressions of generalised moments used in single-site rainfall models* (see above).

`elmstats.dat`: ASCII-format data file containing fitting statistics from an hourly raingauge at Elmdon, near Birmingham in the UK. These data are used in Section 4 below.

`doc_eg.r`: R script containing the commands required to run the examples in Section 4.

`fit_demo.r`: Specimen script to fit models and save the results to file. See end of Section 4.

[1] Note, however, that there is an error on page 17 of Report no. 7: the claim that the weights should not depend on the data is incorrect. This is because weights computed from the data are essentially estimates of the weights corresponding to a single value of $\boldsymbol{\theta}$. We hope to produce a paper in the near future, setting out the correct theory clearly and unambiguously ...

3 Routines provided

The software consists of a suite of R functions. However, only two of the functions are designed for user interaction, so only these are documented in detail here. The remainder are summarised in Appendix A. The function headers in `momfit.r` give more details.

This section is primarily for reference use. Examples illustrating the use of the routines can be found in Section 4 below.

3.1 `auto.fit`

This is the main fitting routine. It is designed to fit a rainfall model to a set of observed rainfall properties, 'automatically' identifying and then exploring promising regions of the parameter space in its search for a global minimum of **(1)**. Optionally, it will then calculate the approximate standard errors and correlation matrix of the parameter estimates, along with objective function thresholds defining approximate confidence intervals for the parameters. The user may impose constraints on the parameters, but this is not necessary.

3.1.1 The algorithm

Typically, the objective function **(1)** has a complicated structure. Experience has shown that it often has local minima, and that poor starting values for an optimisation can result in the failure of standard gradient-based methods. However, given good starting values, gradient methods are usually more powerful than non-gradient methods. For this reason, the basic idea implemented here is to start by using a robust optimisation algorithm (the Nelder-Mead simplex method) to identify promising regions of the parameter space. Having done this, the gradient-based method implemented in function `nlm` is used to refine the optimisation. The structure is as follows:

1. Perform M separate Nelder-Mead optimisations of **(1)**, each beginning from a different initial value of $\boldsymbol{\theta}$. One of these initial values is supplied by the user; the remaining $M-1$ are generated via random perturbations about this user-supplied value.

2. Use a set of heuristics (described below) to update, if possible, the constraints on each parameter, based on the 'best' m parameter sets found so far (i.e. those with smallest objective function values).

3. Discard all but the m best parameter sets found so far. Take the single best set as a new starting point for future optimisations.

4. Use the `nlm` function to perform $N-m$ additional optimisations of **(1)**, each beginning from a different initial value of $\boldsymbol{\theta}$, generated via random perturbations about the starting point identified in step 3.

5. Discard any parameter sets for which the objective function exceeds $10S_m$, where S_m is the mth smallest of the objective function values found in step 2.

6. Repeat steps 2 to 5 a fixed number of times.

In implementing this scheme, the user needs to choose M in step 1, m in steps 2 and 3, N in step 4, and the total number of iterations of steps 2–5. Larger values give an increased chance of finding a global minimum but at the expense of increased computation time. Our experience indicates that setting $M = 100$, $m = M/5$ and $N = 5m/2$, with 5 iterations of steps 2–5, is sufficient to obtain a good estimate of the minimum. These settings are implemented as default values in `auto.fit`, but can be changed by the user.

The heuristics in step 2 are designed to encapsulate the idea that if all of the best parameter sets found so far have similar values of one or more parameters, subsequent searches should be restricted to a neighbourhood of these values. At the same time this neighbourhood should be defined in such a way as to avoid, as far as possible, becoming trapped in a local minimum. The specific details are as follows:

- Define any parameter set that gives an objective function below $10S_m$ (defined above) to be 'promising'. Calculate the largest and smallest values of each parameter among these promising sets. Also calculate the 10th and 90th percentiles of each parameter.
- For each parameter, let U and L be respectively the largest and smallest values among the m best parameter sets found so far, and let C_u and C_ℓ be any existing constraints on the parameter. Subject to the caveats below:
  - If U is less than the 90th percentile just calculated and L is greater than the 10th, constrain the next search to be between $L - (U - L)/10$ and $U + (U - L)/10$.
  - Otherwise, if $C_\ell > -\infty$ redefine it to $L - (C_u - C_\ell)^2/(1 + L - C_\ell)$; similarly, if $C_u < \infty$ redefine it to $U + (C_u - C_\ell)^2/(1 + C_u - U)$. These constraints are somewhat arbitrary, but have the effect of widening the interval (C_ℓ, C_u) if L and U are close to its respective endpoints, and narrowing it otherwise (unless the interval is currently infinite or semi-infinite, in which case it does not make sense to narrow it since we presumably lack sufficient information about the location of the minimum).

  The caveats here are:

  - Never go outside any user-supplied constraints on parameters.
  - Do not make *any* adjustments to existing constraints, without at least 5 'promising' parameter sets.

The R optimisation routines used here are for unconstrained minimisation. In this software, constraints are incorporated by transforming each of the parameters prior to optimisation, so that an unconstrained optimisation can be carried out in the transformed parameter space. The transformations used are:

1. For constraints of the form $\theta > C_\ell$, $\theta^* = \log[\theta - C_\ell]$. The inverse transformation is $\theta = C_\ell + \exp[\theta^*]$.

2. For constraints of the form $\theta < C_u$, $\theta^* = \log[C_u - \theta]$. The inverse transformation is $\theta = C_u - \exp[\theta^*]$.

3. For constraints of the form $C_\ell < \theta < C_u$, $\theta^* = \log\{\log[C_u - C_\ell] - \log[\theta - C_\ell]\}$. The inverse transformation is $\theta = C_\ell + (C_u - C_\ell)\exp\{-\exp[\theta^*]\}$.

Finally: in generating random perturbations about an initial parameter value, the distribution used for each parameter depends on the nature of the constraints for that parameter. For 1-sided constraints, an exponential distribution is used, with expectation equal to the initial parameter value. For 2-sided constraints, a uniform distribution is used. Unconstrained parameters are currently not accommodated, since all parameters appearing in the models considered here are subject to at least one constraint (almost all parameters are strictly positive-valued, for example).

3.1.2 Arguments

The arguments to the function are as follows:

`model.name`: A character string identifying the model to fit. Possible values are:

- `"PO"` Rectangular-pulse Poisson model (Rodriguez-Iturbe et al., 1987).
- `"BL0"` Rectangular-pulse Bartlett-Lewis model (report 8, Section 2).
- `"BL1"` Linear random-parameter Bartlett-Lewis model (report 8, Section 5).
- `"BL2"` Bartlett-Lewis model with 2 cell types (report 8, Section 4).
- `"BL3"` Bartlett-Lewis model with cell depth distribution dependent on duration. This is the simplified version of the model considered in report 8, Section 3, with $d = 2c$ and $g = 2f^2$ in the notation of that report.
- `"QA"` Quadratic random-parameter Bartlett-Lewis model (report 8, Section 6).
- `"NS0"` Rectangular-pulse Neyman-Scott model (Rodriguez-Iturbe et al., 1987). The number of cells in a storm, C say, is such that $C - 1$ has a Poisson distribution. This follows Cowpertwait (1991).
- `"NS1"` Random-parameter Neyman-Scott model. This is as defined in Entekhabi et al. (1989), except that $C - 1$ has a Poisson distribution again.

`theta`: Initial guess at the value of $\boldsymbol{\theta}$ minimising **(1)**. The interpretation of `theta` will vary between models. Table 1 gives details of the model parameterisations. **NB** however: as originally written, the software did not accommodate the use of skewness as a fitting property. In this case, the precise specification of cell intensity distribution

Model	Parameter vector
`"PO"`	$\boldsymbol{\theta} = (\lambda, \mu_X, \sigma_X/\mu_X, \mu_L)'$
`"BL0"`	$\boldsymbol{\theta} = (\lambda, \mu_X, \sigma_X/\mu_X, \beta, \gamma, \eta)'$
`"BL1"`	$\boldsymbol{\theta} = (\lambda, \mu_X, \sigma_X/\mu_X, \alpha, \alpha/\nu, \kappa, \phi)'$
`"BL2"`	$\boldsymbol{\theta} = (\lambda, \mu_{X_1}, \mu_{X_2} - \mu_{X_1}, \sigma_{X_1}, \sigma_{X_2}, 1/\eta_1, 1/\eta_2, \psi_1, \mu_c, \delta_s)'$
`"BL3"`	$\boldsymbol{\theta} = \left(\lambda, \mu_X, \mu_{X\|0}/\mu_X, \delta_c, \mu_c, \delta_s\right)'$
`"QA"`	$\boldsymbol{\theta} = (\lambda, \mu_X, \mu_{X^2}, \alpha, \nu, \kappa_1, \kappa_2, \phi)'$
`"NS0"`	$\boldsymbol{\theta} = (\lambda, \mu_X, \sigma_X/\mu_X, \mu_C, \beta, \eta)'$
`"NS1"`	$\boldsymbol{\theta} = (\lambda, \mu_X, \sigma_X/\mu_X, \alpha, \alpha/\nu, \mu_C, \beta)'$

Table 1: Model parameterisations used in `auto.fit` routine. See relevant sections of report 8 for parameter interpretation. The entries in the "Model" column correspond to values of the function argument `model.name`; those in the "Parameter vector" column correspond to values of the argument `theta`.

is unimportant: the mean μ_X and standard deviation σ_X are all that is required to establish the first- and second-order properties of the model. The choice of distribution makes a difference, however, if skewness is included. The inclusion of skewness is currently incorporated only for the Poisson model `PO`: the expressions programmed here are for a gamma cell intensity distribution.

`timescales`: A list, identifying the fitting properties to be used in the objective function (i.e. the elements of $\mathbf{T}$). The list may contain up to five named components, as follows:

`Mean`: A vector containing the time scales corresponding to any components of $\mathbf{T}$ that are mean rainfall amounts.

`Var`: A vector containing the time scales corresponding to any components of $\mathbf{T}$ that are variances of rainfall amounts.

`Ac`: A 2-dimensional array (or matrix, or data frame) identifying the time scales and lags corresponding to any components of $\mathbf{T}$ that are autocorrelations of rainfall amount time series. The first column contains the time scale and the second the lag.

`Ph`: A vector containing the time scales corresponding to any components of $\mathbf{T}$ that are proportions of wet intervals.

`Skew`: A vector containing the time scales corresponding to any components of $\mathbf{T}$ that are skewness coefficients of rainfall amounts.

`observed`: A vector of observed summary statistics (i.e. the observed value of $\mathbf{T}$), at the time scales defined in `timescales`. The first element(s) must correspond to the mean(s); next variance(s), followed by autocorrelation(s), proportion(s) wet and finally skewness.[2] For example, if `timescales$Mean` is defined as `c(1)` and the time units are hours, the first element of `observed` should be the mean hourly rainfall and the second element should be the variance at a time scale corresponding to the first element of `timescales$Var` (if present — if not, the first element of `timescales$Ac`, `timescales$Ph` or `timescales$Skew`). Note that *the software does not check that the ordering of* `observed` *is correct.*

`covmat`: Optional: the estimated covariance matrix of the statistics in `observed`. This is the matrix $\widehat{\mathrm{var}}(\mathbf{T})$ of report 7, page 15. If present, it can be used to calculate approximate standard errors for the parameter estimates.

`w`: Either

- A $k \times k$ matrix corresponding to the matrix $\mathbf{W}$ in **(1)**, where k is the number of properties used in the fitting procedure (i.e. the length of the `observed` vector); or
- A vector of length k. In this case, the matrix $\mathbf{W}$ in **(1)** is taken to be diagonal and `w` gives the diagonal elements. Thus the parameters are estimated to minimise a weighted sum of squares as in **(2)**.

Defaults to a vector of ones, so that fitting is done by minimising a sum of squares with each property receiving the same weight.

`log.pars`: A logical scalar or vector, indicating whether to optimise over the (natural) logarithms of some parameters. For parameters where this is `TRUE`, the routine transforms the parameter constraints accordingly, computes standard errors etc. on the log scale *and produces all output on the log scale.* The default is `FALSE`.

`log.props`: A logical scalar or vector, indicating which (if any) of the values in `observed` are actually estimates of the *logarithms* of the corresponding properties. The default is `FALSE`. **NOTE:** if a sample statistic T is unbiased for a property τ, then $\log T$ is *not* unbiased for $\log \tau$ — therefore you should *not* change the default setting here unless you are absolutely sure of what you're doing!

`stderr`: Logical scalar determining whether or not to calculate approximate standard errors and confidence limits for the objective function. Defaults to `TRUE` if `covmat` is supplied and `FALSE` otherwise.

[2] Given data $x_1, \ldots, x_n$, the sample skewness is defined as $s^{-3} n^{-1} \sum_{i=1}^{n} (x_i - \bar{x})^3$, where $\bar{x}$ is the sample mean and s is the sample standard deviation.

`numeric.hess`: Logical scalar used if `stderr` is `TRUE`. In this case, the calculation of standard errors etc. requires the expected Hessian of the objective function at the optimum. If `numeric.hess` is `TRUE`, this Hessian will be computed numerically. Otherwise the approximation **(3)** will be used. Since this approximation has slightly better numerical properties, `numeric.hess` defaults to `FALSE`. Note, however, that numerical instabilities can indicate that the optimisation algorithm has failed to converge to a well-defined minimum. For debugging and tracing purposes therefore, it may be helpful to set this argument to `TRUE`. Note also that the use of an analytical approximation does not entirely eliminate the use of numerical differentiation: the gradient vector $\partial\boldsymbol{\tau}/\partial\boldsymbol{\theta}$ is still evaluated numerically, for example.

`conf.level`: A vector of probabilities from which, if `stderr=TRUE`, objective function thresholds will be calculated corresponding to approximate confidence regions for the entire parameter vector. The default value of `c(0.95,0.99)` returns thresholds corresponding to approximate 95% and 99% regions.

`print.level`: Controls the amount of output written to screen during fitting. A value of 0 causes the routine to work silently. A value of 1 (the default) will print a message each time a new set of parameter constraints is calculated (i.e. at the end of step 2 of the fitting algorithm above). Successively higher values yield successively more verbose output.

`lb`: A vector of lower bounds on the parameters in `theta`. The default value is 10^{-6} for every parameter, since most models are parameterised using non-negative quantities (values of exactly zero should be avoided since they can cause pathological problems).

`ub`: A vector of upper bounds on the parameters in `theta`. The default value is ∞ for every parameter.

`ninit`: The number of initial Nelder-Mead optimisations to carry out in step 1 of the fitting algorithm (i.e. the value of M). Default 100.

`nkeep`: The number of parameter sets to keep in step 3 of the algorithm (i.e. the value of m). Defaults to `max(1,0.2*ninit)`.

`nsubseq`: The total number of parameter sets required after step 4 of the algorithm (i.e. the value of N). Defaults to `2.5*nkeep`.

`nsearches`: The total number of times to repeat steps 2–5 of the algorithm. Default 5.

`reltol`: Relative tolerance: if the upper and lower limits on all parameters agree to a relative tolerance of `reltol`, a solution is deemed to have been found and the algorithm will terminate. Default 0.01.

`plot.progress`: A logical scalar indicating whether to plot the objective function values versus the parameters each time the algorithm reaches the end of step 5. This can be useful to monitor the progress of the fitting. Defaults to `TRUE`.

The only arguments that are *required* are `model.name`, `theta`, `timescales` and `observed`.

3.1.3 Value

The function returns a list with the following components:

`Theta`: The optimal parameter vector found. For elements to which a log transformation has been applied using `log.pars`, estimates are returned on a log scale, and this is reflected in the `names` attribute of the result.

`Std.err`: If requested, a vector of approximate standard errors for each of the parameter estimates.

`Corr`: If standard errors were requested, the approximate correlation matrix of the parameter estimates.

`Cov`: If standard errors were requested, the approximate covariance matrix of the parameter estimates.

`Obj`: The value of the objective function at the optimal parameter vector.

`Obj.thresh`: If standard errors were requested, values of the objective function that define an approximate confidence region for the parameter vector, at the levels specified in `conf.level`.

`Gradient`: The gradient vector of the objective function at the optimal parameter vector (obtained via numerical differentiation).

`Hessian`: The Hessian matrix (i.e. matrix of second derivatives) of the objective function at the optimal parameter vector (obtained via numerical differentiation).

`fits.table`: A table containing the final points of other optimisations whose performance is in some sense close to optimal. These are all the parameter sets remaining after the final iteration of step 5 of the fitting algorithm above. The first $2p$ columns of the table (where p is the number of parameters estimated) contain parameter estimates and objective function gradients. The final 3 columns are `Obj`, the objective function, `Method`, the minimisation method that gave rise to this parameter set (`Nelder-Mead` or `nlm`) and `Converge` (a flag indicating the results of the internal convergence checks in R).

`Bounds`: The final set of parameter constraints identified in step 2 of the algorithm.

`Model`: The value of `model.name` on input.

`Timescales`: The value of `timescales` on input.

`Observed`: The value of `observed` on input

`w`: The weights used in the objective function

`log.pars`: A logical vector indicating the parameters to which a log transformation has been applied.

`log.props`: A logical vector indicating the components of `observed` which estimate the logarithms of the corresponding fitting properties.

3.2 `obj.profile`

This function calculates, and optionally plots, profile objective functions for selected parameters. These are obtained by holding the selected parameter fixed at each of a range of values, and optimising over the remaining parameters.

3.2.1 Arguments

`fit`: The output of a call to `auto.fit`, with a non-null value of `Cov`. This will be the case if standard errors were requested in the call to `auto.fit` (see documentation for `auto.fit` for details).

`params`: Numeric vector of parameter for which profiles are required. For example, setting `params=c(1,3)` would calculate profiles for the first and third parameters in `fit$Theta`.

`grids`: An optional list of vectors containing, for each requested parameter, a grid of values at which to evaluate the profile objective function. If `NULL` (the default), a regular grid between `loplim` and `upplim` (see below) is used. **NOTE** that even if some of the parameters in `fits` are on a log scale, all elements of `grids` should be on the *original* scale.

`loplim`, `upplim`: Vectors of lower and upper plotting limits for each requested parameter. Default to estimate $\pm$ 3 standard errors respectively. If supplied by the user, these should be on the original (not logarithmic) scale for all parameters, regardless of whether `log.pars` has been used. These are only used if `grids` is `NULL`.

`lower`, `upper`: Vectors of lower and upper bounds on all model parameters, to constrain optimisation when calculating the objective function profiles. These should all be on the original parameter scale. Defaults are the same as for `auto.fit`.

`npoints`: Scalar giving the number of intermediate points between `loplim` and `loplim` at which to calculate the profile objective function. This is only used if `grids` is `NULL`.

`conf.level`: A vector of probabilities from which objective function thresholds will be calculated corresponding to approximate confidence intervals for individual parameters. The default value of `c(0.95,0.99)` returns thresholds corresponding to approximate 95% and 99% intervals.

`plot.it`: Logical scalar, determining whether or not to plot the results as we go. Default `TRUE`.

`hlines`: Numeric vector, selecting which of the thresholds calculated from `conf.level` to plot as horizontal lines. Defaults to all of them.

`lintyps`: Numeric vector of line types for the various thresholds selected in `hlines`. Defaults to `1+(1:length(hlines))`.

`print.level`: Controls verbosity of output. The function works silently if `print.level=0`; increasing values generate more detailed on-screen progress reports. Defaults to 1.

`ninit`, `nsubseq`, `nsearches`: Arguments to `auto.fit`, when it is called to minimise the objective function at each stage. Default to 2, 10 and 1 respectively (**NB** these differ from the default values in `auto.fit` itself, because when calculating profile objective functions, at each stage we should have a reasonably good idea of where the optimum is located). See `auto.fit` documentation for full details

3.2.2 Value

This function returns a list containing a component for each parameter over which profile objective functions have been calculated. The `names` attribute of the list is `names(fit$Theta)[params]`. Each component is a data frame containing, for each fixed value of the parameter of interest, the optimum values of all the other parameters, the associated objective function values and the thresholds defining profile confidence intervals at the levels specified in `conf.level`. These data frames can be used, for example, to explore the relationships between parameters.

4 Using the software

In this section we illustrate the use of the software with a short demonstration session, using data from an hourly raingauge at Elmdon, near Birmingham in the UK. The data are supplied with the software, in file `elmstats.dat`.

Start up R and change the working directory, if necessary, to that containing the fitting software and specimen data file. Then load the fitting software:

```
> source("momfit.r")
```

Windows users should press `Ctrl-W` at this point, to prevent R from using buffered output (the default behaviour under Windows is to withhold output from the screen until a batch of commands has finished processing — which is not very helpful if you want to watch what is happening!).

Next, read the data into a data frame called `elmdon.data`:

```
 > elmdon.data <- read.table("elmstats.dat",header=TRUE)
```

The data consist of monthly time series of various summary statistics, starting in January 1950 and finishing in September 1997. The data are sorted by month, and within that by year. Display the first line of the data frame:

```
> elmdon.data[1,]
  Month Year Mean1  Var1  Var6 Var24   Ac1 Ac24 Pdry1 Pdry24
1     1 1950 0.019 0.012 0.199 1.366 0.579 0.09 0.941  0.645
```

The column headings are self-explanatory — after the month and year are mean hourly rainfall , variances of 1-, 6- and 24-hourly rainfall, lag 1 autocorrelations of hourly and daily rainfall, and proportions of dry hours and days. Except for the last two columns, these summary statistics will be used to fit a rainfall model. Recall from page 10 that the software uses the proportion of *wet* intervals as a fitting statistic rather than the proportion dry. It is probably easiest simply to replace and rename the existing columns:

```
> elmdon.data[,9:10] <- 1-elmdon.data[,9:10]
> names(elmdon.data)[9:10] <- c("Pwet1","Pwet24")
> elmdon.data[1,]
  Month Year Mean1  Var1  Var6 Var24   Ac1 Ac24 Pwet1 Pwet24
1     1 1950 0.019 0.012 0.199 1.366 0.579 0.09 0.059  0.355
```

To use the fitting routines, we need to make a *single* vector of summary statistics, and to define these to the system using a `timescales` list. Let's obtain summary statistics for January:

```
> mean.stats <- colMeans(elmdon.data[elmdon.data$Month == 1,-c(1,2)],na.rm=TRUE)
> timescales <- list(Mean=1,Var=c(1,6,24),
+ Ac=matrix(c(1,24,1,1),nrow=2),Ph=c(1,24))
> mean.stats
      Mean1        Var1        Var6       Var24         Ac1        Ac24
 0.07993617  0.11353191  1.98474468 13.05380851  0.58904255  0.08402128
      Pwet1      Pwet24
 0.12617021  0.57576596
> timescales
$Mean
[1] 1

$Var
[1]  1  6 24
```

```
$Ac
     [,1] [,2]
[1,]    1    1
[2,]   24    1

$Ph
[1]  1 24
```

The calculation of `mean.stats` discards the first and second columns of `elmdon.data` (which are `Month` and `Year`), and removes missing values. The elements of `timescales` give, in order, the time scale of each of the summary statistics in `mean.stats`. For the `Ac` component, the first column is the time scale and the second is the lag (both autocorrelations are at lag 1 in this case).

If we want to calculate standard errors later, we will need to estimate the covariance matrix of `mean.stats`. This can be done using

$$\frac{1}{n\,(n-1)} \sum_{i=1}^{n} \left(\mathbf{T}_i - \overline{\mathbf{T}}\right) \left(\mathbf{T}_i - \overline{\mathbf{T}}\right)' \ , \tag{4}$$

where $\mathbf{T}_i$ is the vector of summary statistics for the ith January, $\overline{\mathbf{T}}$ is the mean vector (i.e. `mean.stats`) and n is the total number of Januaries in the dataset. The $n-1$ in the denominator is the standard divisor for estimating covariance matrices; the additional factor of n appears because we are estimating the covariance matrix of the *mean* summary statistics $\overline{\mathbf{T}}$, rather than those for an individual January. To implement this in R:

```
> nyears <- sum(!is.na(elmdon.data$Mean1[elmdon.data$Month == 1]))
> var.stats <- var(elmdon.data[elmdon.data$Month == 1,-c(1,2)],na.rm=TRUE)
> var.stats <- var.stats/nyears
```

Now we have some summary statistics and an estimate of their covariance matrix, and can start fitting models. We'll start with a simple 6-parameter Bartlett-Lewis model (report 8, Section 2). From table 1, the parameters for this model are λ (storm arrival rate), μ_X (mean cell intensity), σ_X/μ_X (coefficient of variation of cell intensity distribution), β (cell arrival rate within storms), γ (parameter of exponential storm duration distribution) and η (parameter of exponential cell duration distribution). Based on historical experience, together with some idea of how long storms are expected to last, we might expect that as a rough order of magnitude, $\lambda \approx 0.01, \mu_X \approx 1, \sigma_X \approx \mu_X, \beta \approx 10, \gamma \approx 0.1$ and $\eta \approx 10$. Let's define these as starting values for optimisation:

```
> theta <- c(0.01,1,1,10,0.1,10)
> names(theta) <- c("lambda","mu[X]","sigma[X]/mu[X]","beta","gamma","eta")
```

It is not necessary to name the parameters, but these names will be used to label the output of fitting routines, which aids interpretation.

We're almost ready to fit the model. By default, `auto.fit` produces plots as it works, showing the best-performing values for each parameter. To see all of these in the same window, set up a 2×3 graphics array:

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

Now we can fit the model and store the results in an object called `fit1`, say. Referring to Section 3.1.2, we see that this model is coded as `"BL0"`:

```
> fit1 <- auto.fit(model.name="BL0",theta=theta,timescales=timescales,
+ observed=mean.stats,covmat=var.stats)
Starting initial search
Initial bounds on parameters:
               Lower Upper
lambda         1e-06   Inf
mu[X]          1e-06   Inf
sigma[X]/mu[X] 1e-06   Inf
beta           1e-06   Inf
gamma          1e-06   Inf
eta            1e-06   Inf
Subsequent search 1
Current optimum:
                     Theta      Lower      Upper
lambda          0.03042027 0.02782068        Inf
mu[X]           3.64670343 0.00000100        Inf
sigma[X]/mu[X]  0.16596079 0.00000100  2.9172520
beta            4.63641465 0.00000100 18.0280794
gamma           0.25517173 0.16946983  0.2666243
eta            21.51599021 0.00000100        Inf
Objective function value: 0.0005994
.
.
.
Warning message:
In auto.fit(model.name = "BL0", theta = theta, timescales = timescales,  :
  Hessian is singular - standard errors will be infinite
```

As the fitting routine progresses and the constraints on parameters are updated, progress is written to screen and graphical displays are produced[3]. On completion (after `Subsequent`

[3]Windows users: if progress is *not* written to screen, R is probably buffering the output. You can check this by clicking on the Misc menu.

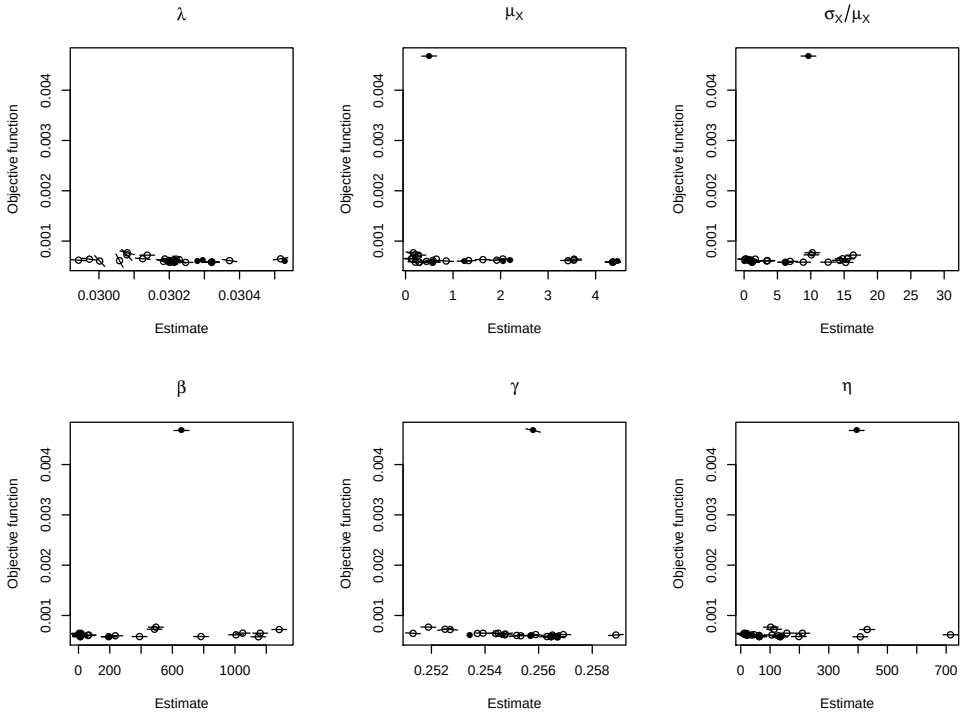

Figure 1: Specimen graphics output from `auto.fit` routine. Lines through the points indicate the corresponding objective function gradients. Filled points are those for which the R internal tests reported successful convergence; open points are those reported as convergence failures.

`search 5`), the graphics window should look something like that in Figure 1. **Note** that the results will not be *exactly* the same, because they depend on the randomly-generated perturbations in steps 1 and 4 of the fitting algorithm. However, if all is well the plots should show clusters of values in similar positions to those in Figure 1, and the objective functions should be similar. The horizontal scales of these plots give some indication of how well-identified the parameters are — for example, λ and γ are tightly constrained whereas the remaining parameters are rather less so.

Warning messages at the end of (and, for some versions of R, during) the fitting are normal. Some arise because the `nlm` routine, which is internal to R, occasionally generates infinite candidate values at which to evaluate the objective function; these should not cause concern. Others result from rounding errors that produce tiny negative results for quantities that should be non-negative (although in general, this should not occur unless the argument `numeric.hess` is explicitly set to `TRUE`). In the example above however, `auto.fit` itself has issued a warning that the standard errors of the parameter estimates are probably infinite. This is quite common for this particular example, and is discussed in more detail below.

The optimal parameter set is now held in the `Theta` component of `fit1`, and the corre-

sponding objective function value is in the `Obj` component:

```
> fit1$Theta
        lambda          mu[X] sigma[X]/mu[X]           beta          gamma
     0.0302466      0.2893516     15.2241771   1149.3796444      0.2567079
           eta
   407.6549382
> fit1$Obj
[1] 0.0005760354
```

Again, the exact values will probably be slightly different but the objective function value, in particular, should be very close to that given above. In our experience, it is common for different — and in some cases *very* different — parameter sets to yield almost identical objective function values. Figure 1, for example, shows that there are near-optimal objective function values for β throughout the range 200 to 1 200, and over a similarly large range of values for η. This suggests that at the parameter estimate there are directions in which the objective function is essentially flat: in this case there can be little information in the objective function about particular parameters or combinations of parameters. A flat objective function has a singular Hessian matrix; we have already seen that the software warned of this. As promised, in this situation it returns infinite standard errors, indicating that (to the accuracy of the numerical optimisation, at least), the full parameter set is not identifiable:

```
> fit1$Std.err
[1] Inf Inf Inf Inf Inf Inf
```

How well does the fitted model reproduce the observed summary statistics? From Appendix A, the function `props.BL0` calculates the expected properties for this model. To find the arguments to this function:

```
> args(props.BL0)
function (theta, timescales)
NULL
```

The function simply requires a parameter vector and a `timescales` object. So, to find the expected properties for the model fitted here:

```
> props.BL0(fit1$Theta,timescales)
$Mean
[1] 0.09614596

$Var
```

```
[1]  0.1038288  1.9868603 13.0534991

$Ac
[1] 0.58800947 0.09080338

$Ph
[1] 0.1377335 0.5699538

$Skew
NULL
```

A comparison of these expected properties with those observed in `mean.stats` reveals a problem. The 24-hour variance is reproduced well by the fitted model, but the 1-hour mean, which is arguably the most important property, is reproduced much less well, with an error in excess of 15%. The problem is partly due to the fact that larger values (such as the 24-hour variance) dominate the objective function. In the rainfall modelling literature, it has become accepted practice therefore to downweight the larger properties so as to reduce their impact on the final result.

In statistical terms however, the relevant consideration is the precision, rather than the magnitude, of the contributions to the objective function. From this point of view, the argument for downweighting the 24-hour variance is not that it dominates the objective function: rather, it is typically estimated much less precisely than (say) the 1-hour mean and therefore, in some sense, carries relatively little information. In general, it may be shown using an extension of the arguments in Hansen (1982) that in large samples, there is an optimal choice of weighting matrix $\mathbf{W}$ in **(1)**: this optimal choice is $\mathbf{W} = \mathbf{\Sigma}^{-1}$, where $\mathbf{\Sigma}$ is the covariance matrix of the fitting properties $\mathbf{T}$. Asymptotically, *any linear combination of the model parameters is estimated at least as precisely for this choice of* $\mathbf{W}$ *as for any other*. This suggests that problems of parameter identifiability can be reduced as far as possible by using this choice of $\mathbf{W}$. The problem, of course, is that $\mathbf{\Sigma}$ is unknown. It can be estimated from the data, however, using an expression such as **(4)**. We have conducted some simulations to determine how well the resulting estimation procedure works in practice. The results are encouraging and indicate that in general, parameters are indeed much better identified than using other weighting schemes that have been suggested in the literature. Unfortunately however, the standard errors reported by the software for this particular choice of $\mathbf{W}$ tend to be *extremely* inaccurate. The reason for this is essentially that **(4)** estimates the $k(k+1)/2$ unique elements of $\mathbf{\Sigma}$ separately, and the individual estimation errors cumulate to swamp the inference about $\boldsymbol{\theta}$ (which typically has far fewer than $k(k+1)/2$ elements).

Work is in progress to alleviate the problem of estimating $\mathbf{\Sigma}$ by way of obtaining an optimal weighting scheme. In the interim however, a reasonable alternative is just to take the diagonal elements of $\mathbf{\Sigma}$ and to minimise a weighted sum of squares of the form **(2)**, with $w_i = 1/\hat{\mathrm{V}}\mathrm{ar}\,(T_i)$. If the fitting properties were known to be mutually uncorrelated, this would provide an estimate of $\mathbf{\Sigma}^{-1}$ and hence would deliver asymptotically optimal estimators. Our simulation experiments indicate that this particular choice of weights can deliver performance

that is almost as good as the optimal choice, but that the standard errors and confidence intervals reported by the software are much more accurate. To calculate this choice of weights therefore:

```
> prop.wt <- 1 / diag(var.stats)
> prop.wt
       Mean1          Var1          Var6         Var24           Ac1          Ac24
3.756028e+04 8.955952e+03 2.494121e+01 3.125249e-01 4.994064e+03 1.819588e+03
       Pwet1       Pwet24
2.761717e+04 2.014010e+03
>
```

Notice that the 24-hour variance now has a very small weight compared to the other properties. This reflects the fact that it is estimated imprecisely, and hence it is inappropriate to try and force too close a match to this property.

Now let's refit the model. This time, we will use the parameter set we have already found as a starting value, and will carry out the optimisation on a log scale:

```
> par(mfrow=c(2,3))
> fit2 <- auto.fit(model.name="BL0",theta=fit1$Theta,timescales=timescales,
+ observed=mean.stats,covmat=var.stats,w=prop.wt,log.pars=TRUE)
Starting initial search
Initial bounds on parameters:
                        Lower Upper
log(lambda)         -13.81551   Inf
log(mu[X])          -13.81551   Inf
log(sigma[X]/mu[X]) -13.81551   Inf
log(beta)           -13.81551   Inf
log(gamma)          -13.81551   Inf
log(eta)            -13.81551   Inf
.
.
.
```

Notice now that the fitting, parameter constraints and graphics for all parameters are on a log scale. You may also notice that the fitting is much quicker — this appears to be true in general for log-transformed parameters. To find the optimal parameter estimates and objective function:

```
> fit2$Theta
 log(lambda)          log(mu[X]) log(sigma[X]/mu[X])           log(beta)
 -3.53043123         -0.07371271         -0.12579069         -0.78523377
  log(gamma)            log(eta)
```

```
 -1.68594283          0.14719832
> fit2$Obj
[1] 1.527799
```

The estimates are given on a log scale. The objective function is not comparable with that obtained previously, because the fitting properties have been reweighted. Let's translate the parameter estimates back to their original scale:

```
> exp(fit2$Theta)
 log(lambda)          log(mu[X]) log(sigma[X]/mu[X])           log(beta)
  0.02929228          0.92893853          0.88179939          0.45601309
  log(gamma)            log(eta)
  0.18526967          1.15858371
```

Ignoring the incorrect variable names here, the estimates of λ and γ are similar to those obtained previously. Those for the other three parameters, however, are rather different. It seems that reweighting the fitting properties has made a substantial difference to the characteristics of the objective function. Nonetheless, the motivation behind the reweighting was to improve the precision of the fit and, since the standard errors under the previous fit were reported as infinite, the new estimates must be consistent with the old ones!

Let's compare the observed and expected properties under the new fit:

```
> props.BL0(exp(fit2$Theta),timescales)
$Mean
[1] 0.0812939

$Var
[1]  0.1197759  1.9304441 11.3129133

$Ac
[1] 0.59091670 0.08524093

$Ph
[1] 0.1267480 0.5703441

$Skew
NULL
```

In terms of agreement between observed and expected properties, this is clearly an improvement upon the original fit; unsurprisingly, the match between observed and expected 24-hour variances is less good now, but given that the standard error of the sample 24-hour variance is about 1.8 (this can be seen from the square roots of the diagonal elements of `var.stats`), the fitted value of 11.3 is easily consistent with the observed value of 13.1.

For this particular choice of weights, the software does not usually issue a warning message about singular Hessians. This suggests that the parameters are now much better identified. To check this we can look at the standard errors:

```
> fit2$Std.err
[1] 0.07704442 0.09085994 0.05373905 0.52816406 0.34058291 0.11053392
```

Note that these standard errors correspond to the log-transformed parameters since the fitting was done on a log scale. The standard errors for the fourth and fifth parameters ($\log\beta$ and $\log\gamma$) therefore correspond to quite large relative uncertainties on the original scale. Overall however, these standard errors are modest in size; this gives some confidence that the revised weights do indeed improve the identifiability of the parameters. A by-product of this is increased numerical stability: the numerical optimisation problem becomes easier, and hence different fitting runs tend to give similar results.

Another way to investigate parameter uncertainty or identifiability is to plot profile objective functions showing the optimum achievable objective function for each value of a particular parameter. The function `obj.func` can be used to produce such plots, and store the results. To plot profiles for λ and μ_X, over the ranges $(0.015, 0.05)$ and $(0.04, 1.5)$ respectively:

```
> par(mfrow=c(1,2))
> prf.tabs <- obj.profile(fit2,params=c(1,2),
+ loplim=c(0.015,0.04),upplim=c(0.05,1.5),npoints=20)
Calculating profile objective function for log(lambda) ...
log(lambda) = -3.5375   Objective function = 1.5378
.
.
.
```

The results will be similar to those shown in Figure 2. The profiles are produced on a log scale, corresponding to the estimates in `fit2`. The horizontal lines on the plots define approximate 95% and 99% confidence intervals for each parameter, as described in report 7. The lines are at different levels for the two parameters — this is to be expected from the theory. λ appears better identified than μ_X — approximate 95% confidence intervals for $\log\lambda$ and $\log\mu_X$ are around $(-3.7, -3.4)$ and $(-0.7, 0.0)$ respectively (this can be verified from the profile objective function values stored in `prf.tabs`), corresponding to intervals of $(0.025, 0.033)$ and $(0.50, 1.0)$ for λ and μ_X themselves. Notice that the profile for μ_X flattens out so that at the 99% level the confidence interval for this parameter is much wider (although the upper end of the interval does not change much).

These examples are intended to give an overview of what can be achieved using this software, with most of its default settings. In practice, users will probably want to write scripts to fit models and save the results to file. An example of such a script is provided in

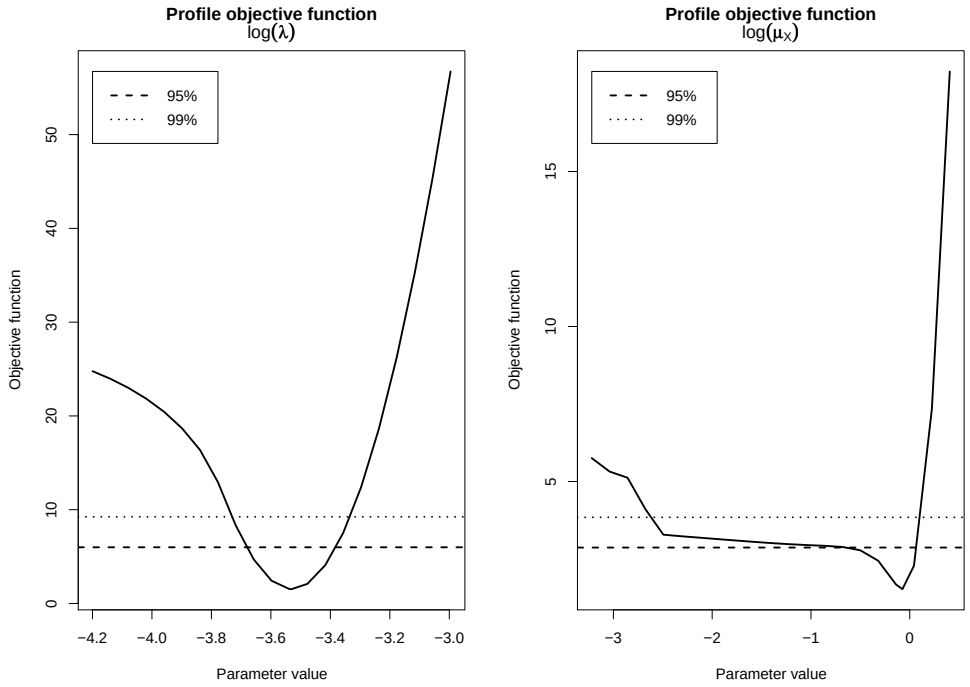

Figure 2: Profile objective functions for $\ln \lambda$ and $\ln \mu_X$, obtained using `obj.profile`. Horizontal lines define approximate confidence intervals at the levels indicated.

file `fit_demo.r`, which is included with the software distribution. This fits separate random-parameter Bartlett-Lewis models to each month of the year at Elmdon, stores the parameter estimates and standard errors, and uses these to produce plots of the seasonal variation in each parameter. See the comments in the script for further details.

5 Bugs and problems

This software is still undergoing development and, in places, should still be regarded as 'experimental'. We are aware of the following problems:

1. The numerical differentiation required to calculate standard errors can be unstable. This may give rise to problems in matrix inversions, or more generally to inaccurate calculation of the standard errors themselves. Our experience is that such problems can be reduced, if not entirely eliminated, by using weights in the objective function that ensure reasonable parameter identifiability as in the `fit2` example above.

2. For some models, the algebraic expressions for certain properties are undefined at specific parameter values. For example, for the linear random-parameter Bartlett-Lewis model `BL1`, a key quantity involves expressions that are undefined at $\alpha = 3$ and

need to be evaluated via an appropriate limiting operation. Such special cases have *not* been implemented in the code provided here.

3. Neither the derivation of properties, or the coding implementation, for the quadratic random parameter model (`QA` in Section 3.1.2) have been checked yet.

4. For the random-parameter Neyman-Scott model `"NS1"`, expressions for second-order properties are taken from Entekhabi et al. (1989). These appear to be based on series expansions that are valid only when $\beta^2 \ll \eta^2$ (which may cause problems as the optimisation algorithm explores the parameter space). Further, the approximations are undefined when the shape parameter, α, of the randomisation distribution takes values 1, 2, 3, 4 or 5. Finally, the expression for Var $\left[Y_t^{(h)}\right]$ given in Entekhabi et al. (1989) looks suspiciously as though it is in fact the expression for E $\left[\left(Y_t^{(h)}\right)^2\right]$. The code implements the expression that is given in the paper (and reproduces the results reported in that paper), but the results should be checked via simulation.

5. At present, we have programmed the theoretical skewness coefficients only for the Poisson model `P0`. Skewness for other models will be added as and when we have a few days spare to write and check the necessary code. Or if any users of these routines feel like sending us (checked and readable!) `R` code for skewness, that is suitable for incorporating into these routines with due acknowledgement.

Any other problems should be reported to Richard Chandler (r.chandler@ucl.ac.uk).

References

Cowpertwait, P. S. P. (1991). Further developments of the Neyman-Scott clustered point process for modeling rainfall. *Water Resources Research*, 27:1431–1438.

Entekhabi, D., Rodriguez-Iturbe, I., and Eagleson, P. S. (1989). Probabilistic representation of the temporal rainfall process by a modified Neyman-Scott rectangular pulses model: parameter estimation and validation. *Water Resources Research*, 25, No.2:295–302.

Hansen, L. R. (1982). Large sample properties of generalized method of moments estimators. *Econometrica*, 50:1029–1054.

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

Rodriguez-Iturbe, I., Cox, D. R., and Isham, V. (1987). Some models for rainfall based on stochastic point processes. *Proc. R. Soc. Lond.*, A410:269–288.

Appendices

A Summary of R functions in file momfit.r

Function	Purpose
auto.fit	See Section 3.1.
grad.fn	Utility function, enabling numerical differentiation of expected properties with respect to parameters.
grad.trans	Utility function, used to deal with parameter transformations when calculating property:parameter derivatives.
I.NS1	Computationally efficient evaluation of the expression $\Gamma(\alpha - x)/\Gamma(\alpha)$, which appears in properties of random-parameter Neyman-Scott model (Entekhabi et al., 1989).
inter	Approximation to the integral required for the proportion of dry periods in the random-parameter Bartlett-Lewis model (report 8, equations 46 and 48).
inter2	Approximation to the integral required for the mean storm duration in a Bartlett-Lewis model (report 8, Section 2.4).
model.select	Call the appropriate props.??? routine to calculate expected properties for a particular model.
moment.fit	Optimisation routine, with options controlling, for example, the optimisation method used. auto.fit calls this routine with options that have been found to work well in practice.
num.deriv	Numerical differentiation (R doesn't have a routine that does this directly). This is an implementation of Ridders' method, closely following Press et al. (1992, §5.7).
num.hess	Numerical evaluation of a Hessian matrix.
objfunc	Compute the weighted sum of squared differences between observed and expected properties, for a particular model and parameter value.
obj.profile	See Section 3.2.
par.trans	Utility function to transform parameters for optimisation.
plot.mmfit	Plot the table of results from a call to moment.fit. This is called automatically by moment.fit and auto.fit, if plots are requested.
props.PO	Calculate expected properties for the rectangular pulse Poisson model (Rodriguez-Iturbe et al., 1987).
props.BL0	Calculate expected properties for the rectangular pulse Bartlett-Lewis model (report 8, §2).

Function	Purpose
`props.BL1`	Calculate expected properties for the linear random-parameter Bartlett-Lewis model (report 8, §5).
`props.BL2`	Calculate expected properties for the 2-cell Bartlett-Lewis model (report 8, §4 with $N = 2$).
`props.BL3`	Calculate expected properties for the dependent depth-duration Bartlett-Lewis model (report 8, §3).
`props.QA`	Calculate expected properties for the quadratic random-parameter Bartlett-Lewis model (report 8, §6).
`props.NS0`	Calculate expected properties for the rectangular pulse Neyman-Scott model (Rodriguez-Iturbe et al., 1987).
`props.NS1`	Calculate expected properties for the random-parameter Neyman-Scott model (Entekhabi et al., 1989).
`pt.ns0`	Integrand in expression for the probability that an interval of length h is dry in the rectangular pulse Neyman Scott model (Cowpertwait, 1991).