

Univariate Frequency Analysis Using **FAmLe**

François Aucoin

June 12, 2017

1 Introduction

FAmLe is a R package that may be used in order to carry out tasks that pertain to univariate frequency analysis. Basically, two general frameworks are being entertained here: *maximum likelihood* and *Bayesian*. More precisely, for a given problem, the user may decide to estimate the unknown parameters via maximum likelihood, and to then proceed to statistical inference accordingly, or to use a full Bayesian approach to parameter estimation and inference.

Although the help files provided with **FAmLe** are regarded as self-contained, such that the user should be able to get the package to work without having to rely upon extra documentation, a careful examination of the present document would nonetheless correspond to a good time investment. In other words, by carefully reading this document and by going through the examples one by one, the user will, besides from being exposed to the functions available in the package, benefit from seeing how those functions are actually being implemented in real problems.

It seems worthwhile, at this point, to inform the potential user of this package about a few facts. First, as **FAmLe** was originally developed in the context of univariate frequency analysis in flood hydrology, many of the examples presented herein are biased towards that specific topic. In fact, not only the examples are biased, but also some of the functions made available by this package. For example, in univariate flood frequency analysis, the main interest often lies in estimating quantiles for some high return periods; the former characterizing extreme flood events that occur with low probabilities. That said, although the main functions described herein (i.e., those outlined in the sections' titles) may be useful for estimating probability distributions regardless of the application field, the extra functions (as they will be presented below), were mostly written in view to tackle flood quantile estimations. Secondly, although, without a single doubt, packages similar (with many respects) to **FAmLe** may already exist, it is believed here that the greatest feature of the latter lies in its generality, with respect to both maximum likelihood estimation and Bayesian estimation of univariate frequency distributions (this goes without saying, however, that other possibly available packages are not general). Thirdly, and finally, the user should understand, before making any statistical inferences (e.g., significance tests, prior elicitation, or confidence intervals), that it is her/his responsibility to make sure that the theoretical arguments behind their use of the functions made available by **FAmLe** are valid. For example, maximum likelihood estimators often benefit from nice asymptotic properties. However, there are conditions that must be met in order for such properties to apply, and, although good references will be given, such conditions will not explicitly be discussed in this document.

This `vignette` is organized as follows. In Section 2, the `distr` function, which may be regarded as the chore of the `FAMle` package, is presented and described thoroughly. In Section 3, the `mle` function, which may be used to fit a parametric probability distribution to a given univariate dataset, is presented and put to work in an application to life-time data. Section 4 briefly discusses the `boot.mle` function, which may be used to generate parametric bootstrap distributions from the fitted parametric model, while Section 5 illustrates how the so-generated bootstrap distributions may be used to derive different types of confidence intervals as well as to carry out two basic composite hypotheses tests concerning the fitted distribution's plausibility in light of the actual dataset. The latter section also treats the topic of approximate normal confidence intervals for some quantile estimates. Section 6 deals with the declaration of new probability distributions, for the purpose of later using them in conjunction with the `FAMle` functions. Section 7 presents the `FAdist` package (Aucoin, 2011), which contains a series probability distributions sometimes used in practice, but which are not available in R. The `metropolis` function, which may be used to carry out a full Bayesian estimation of the unknown parameters, is introduced and put to work in Section 8.

2 The `distr` Function

As mentioned in Section 1, `distr` corresponds to the most important `FAMle` function. The reason for this is that `distr` is being called (and thus required) by most of the remaining functions to perform their respective tasks. It thus seems natural to begin this tutorial by presenting `distr` and by giving several simple examples on how it may be used. Before going any further, however, the user should first make sure that the `FAMle` package is loaded (by running the following code in R), and then carefully read the help file available for `distr`:

```
> library(FAMle)
> help(distr)
```

As may be understood from the help file discussed at the previous paragraph, `distr` is a function that allows to call any of the four functions that pertain to each one of the univariate probability distributions already available in R. For example, for the well-known normal distribution, the following four functions are available: `dnorm`, `pnorm`, `qnorm`, and `rnorm`. Basically, these functions, when called and provided with the proper arguments, will respectively evaluate (and/or return) the following: the density function, the cumulative distribution function, the quantile function, and random variates. A numerical example shall make everything more obvious!

```
> x <- -4:4
> x
```

```
[1] -4 -3 -2 -1  0  1  2  3  4
```

```
> dnorm(x=x,mean=0,sd=1)
```

```
[1] 0.0001338302 0.0044318484 0.0539909665 0.2419707245 0.3989422804
[6] 0.2419707245 0.0539909665 0.0044318484 0.0001338302
```

```

> Fx <- pnorm(q=x,mean=0,sd=1)
> Fx

[1] 3.167124e-05 1.349898e-03 2.275013e-02 1.586553e-01 5.000000e-01
[6] 8.413447e-01 9.772499e-01 9.986501e-01 9.999683e-01

> qnorm(p=Fx,mean=0,sd=1)

[1] -4 -3 -2 -1 0 1 2 3 4

> set.seed(123)
> rnorm(length(x),mean=0,sd=1)

[1] -0.56047565 -0.23017749 1.55870831 0.07050839 0.12928774 1.71506499
[7] 0.46091621 -1.26506123 -0.68685285

```

The next code chunk illustrates how exactly the same tasks may be performed using `distr`:

```

> distr(x=x,dist='norm',param=c(0,1),type='d')

[1] 0.0001338302 0.0044318484 0.0539909665 0.2419707245 0.3989422804
[6] 0.2419707245 0.0539909665 0.0044318484 0.0001338302

> Fx <- distr(x=x,dist='norm',param=c(0,1),type='p')
> Fx

[1] 3.167124e-05 1.349898e-03 2.275013e-02 1.586553e-01 5.000000e-01
[6] 8.413447e-01 9.772499e-01 9.986501e-01 9.999683e-01

> distr(x=Fx,dist='norm',param=c(0,1),type='q')

[1] -4 -3 -2 -1 0 1 2 3 4

> set.seed(123)
> distr(x=length(x),dist='norm',param=c(0,1),type='r')

[1] -0.56047565 -0.23017749 1.55870831 0.07050839 0.12928774 1.71506499
[7] 0.46091621 -1.26506123 -0.68685285

```

In fact, there are other arguments that may be provided to some of the four functions that pertain to the normal distribution (this is also true for all distributions available in R): `log`, `lower.tail`, `log.p` (see `help(norm)` for more information). For instance, it is often useful to evaluate the log-density for a specific vector of observations. As an example:

```

> dnorm(x=x,mean=0,sd=1,log=TRUE)

[1] -8.9189385 -5.4189385 -2.9189385 -1.4189385 -0.9189385 -1.4189385 -2.9189385
[8] -5.4189385 -8.9189385

> distr(x=x,dist='norm',param=c(0,1),type='d',log=TRUE)

[1] -8.9189385 -5.4189385 -2.9189385 -1.4189385 -0.9189385 -1.4189385 -2.9189385
[8] -5.4189385 -8.9189385

```

In other situations, it is convenient to evaluate a function for various parameter values. For example, given several pairs of location and scale parameters, one may wish to compute the p -th quantile of a normal distribution. Here is how this may be done using both `qnorm` and `distr`:

```

> p <- .95
> mu <- c(0,1,2)
> s <- c(.1,.2,.3)
> cbind(R=qnorm(p=p,mean=mu,sd=s),FAmle=distr(x=p,dist='norm',
+   param=cbind(mu,s),type='q'))

           R      FAmle
[1,] 0.1644854 0.1644854
[2,] 1.3289707 1.3289707
[3,] 2.4934561 2.4934561

```

From the previous example, it can be seen that the `param` argument may be specified (row-wise) as an array. In fact, it would also be possible to provide a probability vector, as long as the latter has length equal to the number of rows for the array of parameters:

```

> p <- c(.1,.5,.9)
> cbind(R=qnorm(p=p,mean=mu,sd=s),FAmle=distr(x=p,dist='norm',
+   param=cbind(mu,s),type='q'))

           R      FAmle
[1,] -0.1281552 -0.1281552
[2,]  1.0000000  1.0000000
[3,]  2.3844655  2.3844655

```

In other words, if `param` is to be specified as an array, the user should then specify `x` as either a scalar (vector of length equal to 1) or a vector that has length equal to the number of rows in `param`.

Finally, the `model` argument of `distr` is not discussed within this section. Understanding of the latter requires that the `mle` function be first understood. That is the topic of the next section!

3 Fitting a Distribution Using `mle`

This section deals with describing the `mle` function, which, as already stated, may be used to fit probability distributions to a given univariate dataset. As pointed out in Section 1, no theoretical development pertaining to maximum likelihood estimators' asymptotic properties will be presented here. For more information on that topic, the interested reader is referred to two classical texts in mathematical statistics, namely, Rice (2003) and Hogg *et al.* (2004), as well as to the references therein. Within a context very similar to that of the present document, Coles (2001) presents a nice review on maximum likelihood theory.

As was suggested for the `distr` function presented at the previous section, it is, at this point, strongly recommended that the user carefully reads the help file pertaining to `mle` before continue reading!

```
> help(mle)
```

The best way to start is by considering a simple example where a specific univariate probability distribution needs to be fitted to a dataset. To such ends, datasets were included with `FAmle` when the latter was built. Information regarding all datasets included with `FAmle` can be accessed as follows:

```
> help(package='FAmle')
> data(package='FAmle')
```

For this first example, the `yarns` dataset, which is used by Gamerman & Lopes (2006) (and references therein), is considered. That dataset first needs to be loaded into R, and this may be done as follows:

```
> data(yarns)
> x <- yarns[,1]
> hist(x, freq=F, col='grey', border='white', main='')
```

Figure 1 shows an histogram of the `yarns` dataset, for which a new variable named `x` was declared in the above code chunk. As stated by Gamerman & Lopes (2006), the Weibull distribution is often regarded as a plausible model for this type of data. Interestingly, functions for the Weibull distribution are available in R, such that it would make sense here to use it for this first example.

```
> fit.x <- mle(x=x, dist='weibull', start=c(.1, .1))
> fit.x
```

```
-----
                Maximum Likelihood Estimates
-----
Data object:  x
Distribution:  weibull

----- Parameter estimates -----
```

```

          shape.hat scale.hat
Estimate  1.6052    247.86
Std.err   0.1219    16.27

----- Goodness-of-Fit -----

log.like      aic      ad      rho
-625.1990 1254.3981  0.5282  0.9836
-----

> class(fit.x)

[1] "mle"

```

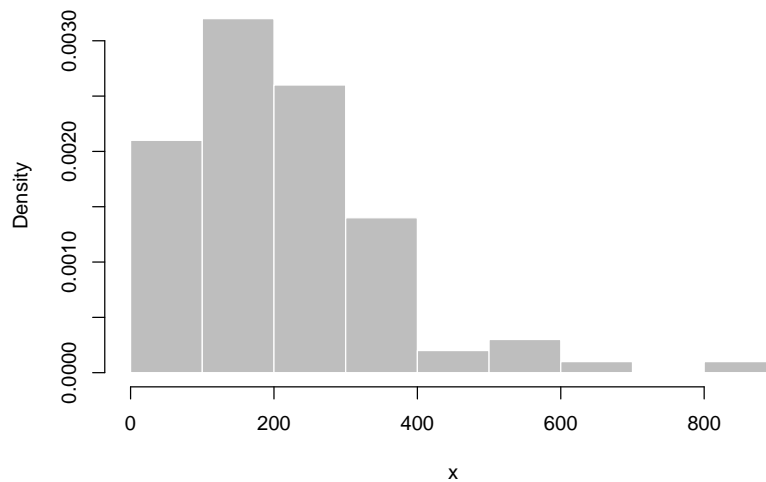


Figure 1: Histogram of 100 cycles-to-failure times for the *airplane yarns* dataset described by Gamerman & Lopes (2006).

Besides showing how fitting a distribution is done using `mle`, the above code chunk also illustrates the output that is returned when calling `print.mle` (note: that `mle` returns an object from the class `mle`, which allows it to work directly with `plot`). It can also be noticed, from the previous output, that some *goodness-of-fit* statistics are returned, in addition to the parameter estimates and their respective estimators' approximated standard errors (see `help(mle)` for more information). As will be shown later in this document, the `ad` and `rho` goodness-of-fit statistics may be used in order to derive composite hypotheses tests which permit to assess the fitted model's plausibility in light of the dataset at hand. For the moment, however, it is still worthwhile, as well as probably more informative, to try assessing the quality of the fitted model using diagnostic plots. Similar to `print.mle`, a

function called `plot.mle` (see `help(plot.mle)`) may be used to generate a series of four diagnostic plots for the `mle` object stored within `fit.x`. For this example, the plots generated by `plot.mle` are presented by Figure 2.

```
> alpha <- .05
> plot(x=fit.x, ci=TRUE, alpha=alpha)
```

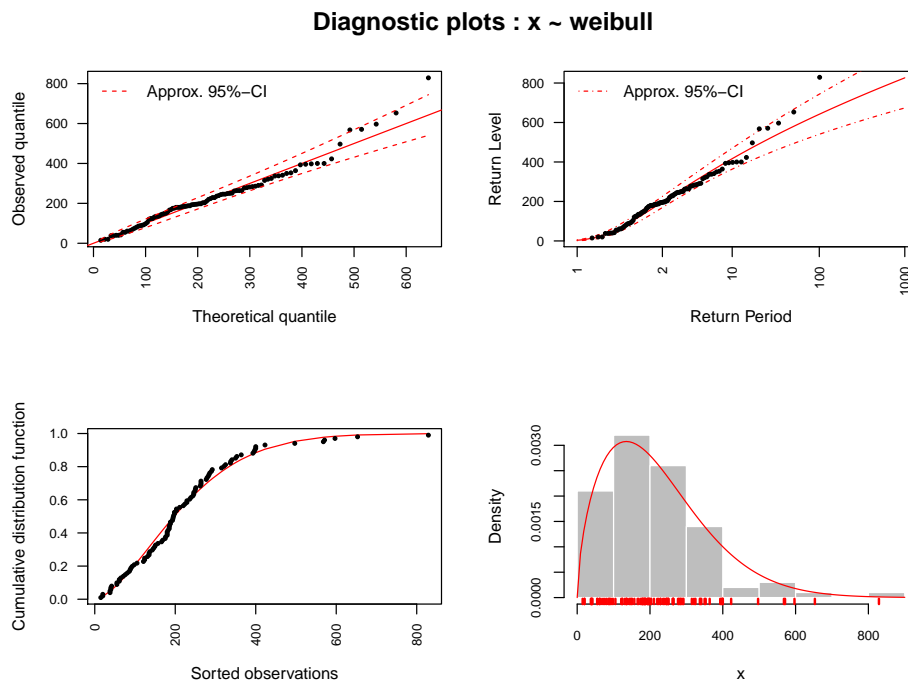


Figure 2: Diagnostic plots for the weibull model fitted to the `yarns` dataset. The dashed red lines correspond to the lower and upper confidence bounds of the approximate 95% confidence intervals derived using the observed Fisher's information matrix in conjunction with the well-known delta method.

From the above code chunk, it can be seen that two arguments, namely, `ci` and `alpha`, were also specified in addition to the `mle` object `fit.x`. For the moment, it only has to be understood that those two arguments are optional and that they were used here to generate the approximate confidence intervals shown by Figure 2. Confidence intervals will be discussed in more detail in Section 5.

At this point, it is also important to note that several elements are contained in the `fit.x` object:

```
> names(fit.x)

[1] "fit"          "x.info"      "dist"        "par.hat"     "cov.hat"     "k"
[7] "n"           "log.like"    "aic"         "ad"          "data.name"   "rho"
```

Obviously, all of these elements are described in detail by the `mle` help file, and that file should again be consulted by the reader before continuing.

In Section 2, the `distr` function was thoroughly described, but the argument `model` was left uncommented. Here, a few additional simple examples are given on how the returned `mle` object stored in `fit.x`, for instance, may be used in conjunction with `distr`. In many practical problems dealing with time-to-failure data, interests often lie in the survival function. For example, for the `yarns` dataset, one might wish to obtain an estimate of the probability of survival above a certain amount of time units. In this case, assuming that $F_X \sim Weibull(\alpha, \beta)$, with $\hat{\alpha}$ and $\hat{\beta}$ corresponding to the maximum likelihood parameter estimates of the Weibull distribution for the `yarns` dataset, what is $\Pr[X \geq 400 | \hat{F}_X]$? For the fitted model, the `distr` function may be used to compute that probability:

```
> x.obs <- 400
> distr(x=x.obs, dist=fit.x$dist, param=fit.x$par.hat, type='p', lower.tail=FALSE)
```

```
[1] 0.1157897
```

```
> pweibull(q=x.obs, shape=fit.x$par.hat[1], fit.x$par.hat[2], lower.tail=FALSE)
```

```
[1] 0.1157897
```

That is, the elements of the fitted model `fit.x` may be used in conjunction with `distr` to compute all sorts of quantities. For a fitted model, however, instead of specifying the arguments `dist` and `param`, one may optionally choose to only specify the fitted model as

```
> distr(x=x.obs, model=fit.x, type='p', lower.tail=FALSE)
```

```
[1] 0.1157897
```

which makes things somewhat faster!

As a second example, once more carried out using the elements available from the fitted model `fit.x`, a quick survival plot, which is illustrated by Figure 3, may be built as follows:

```
> sorted.S.x <- 1 - fit.x[['x.info']][, 'Fz']
> sorted.x <- fit.x[['x.info']][, 'z']
> empirical.S.x <- 1 - fit.x[['x.info']][, 'Emp']
> plot(sorted.x, sorted.S.x, type='l', col='red',
+       xlab=expression(x), ylab=expression(S(x)==1-F(x)))
> points(sorted.x, empirical.S.x, pch=19, cex=.5)
```

In addition to the latter, Figure 3 also shows an histogram of the `yarns` dataset, along with the fitted density curve for model `fit.x`. That second plot may be built as follows:

```
> hist(x, freq=FALSE, main='', col='cyan')
> fun.x <- function(x) distr(x=x, model=fit.x, type='d')
> curve(fun.x, add=T, col='red')
```

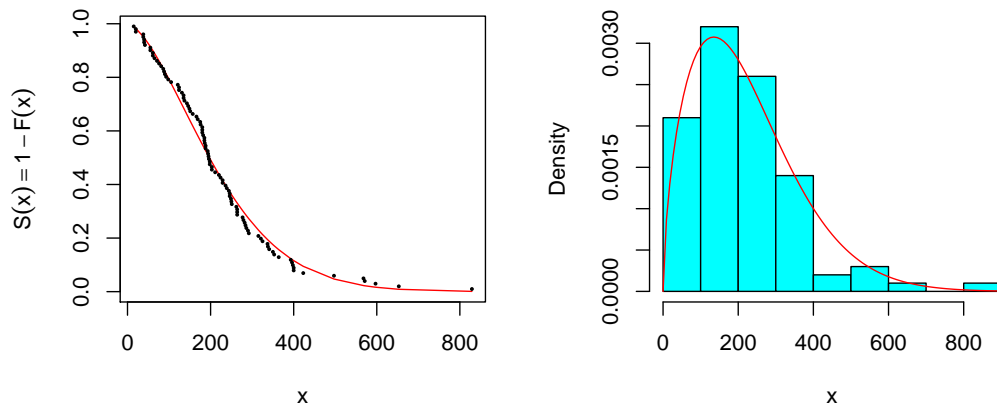



Figure 3: (a) Survival plot and (b) histogram with fitted density curve, both obtained using the fitted model `fit.x`.

This section ends with an example in which several probability distributions are fitted to a same dataset, and where the goodness-of-fit statistics are used as a means of assessing which model might be the best. For instance, in their example using the `yarns` dataset, Gamerman & Lopes (2006) (page 255) consider three models: Weibull, lognormal, and gamma. As those distributions are all available in `R`, the following code chunk illustrates how they may be simultaneously fitted to the dataset at hand.

```
> dist.vec <- c('weibull', 'lnorm', 'gamma')
> fit.3 <- list()
> for(i in dist.vec) fit.3[[i]] <- mle(x=x, dist=i, start=c(.1, .1))
```

Assuming that only `ad` and `aic` are of interest for the moment, the following code chunk illustrates how such statistics may be extracted from the `fit.3` object.

```
> sapply(fit.3, function(h) c(ad=h$ad, aic=h$aic))
```

	weibull	lnorm	gamma
ad	0.5282333	2.011448	0.6646458
aic	1254.3980555	1267.520669	1254.4886833

Usually, smaller values of both `ad` and `aic` suggest a potentially better fit. Obviously, used this way, those values do not actually tell anything about whether or not the fits seem appropriate, but only which one is best in comparison to the others. In this case, both statistics suggest that Weibull should be preferred. In fact, there often exists a theoretical argument behind the use of the Weibull distribution when dealing with this type of data.

At this point, the reader may have noticed that the maximum likelihood estimates for the lognormal distribution were obtained iteratively even though a closed form solution does

exists for the maximum likelihood parameter estimators of this distribution. The reason for this is simple: generality. In other words, the `FAmle` package was written to be as general as possible. Obviously, when aiming for generality, there must always be a trade-off between being general and computationally efficient. This is an example for which efficient computing was sacrificed for convenience. Other examples like this one will be encountered in subsequent sections (although they will not necessarily be explicitly pointed out).

4 Parametric Bootstrap Using `boot.mle`

In this section, the `boot.mle` function will be briefly described and used to generate parametric bootstrap parameter samples for some fitted model – Davison & Hinkley (1997) is an excellent book about bootstrap methods. The `yarns` dataset example is now put aside, and a new dataset is considered. That new dataset consists of annual maximum daily average flows (in m^3/s) that were observed at a particular hydro-metric station located along a river in New Brunswick (Canada), and may be loaded into R as follows:

```
> data(station01AJ010)
> y <- station01AJ010
> layout(matrix(1:2,nr=1))
> plot(as.numeric(names(y)),y,type='b',cex=.4,pch=19,
+      cex.axis=.75,xlab='Year',ylab='y')
> lines(as.numeric(names(y)),lowess(y)[['y']],col='red')
> legend('topleft',inset=.01,col='red',lwd=2,'Lowess',bty='n')
> acf(y,main='')
```

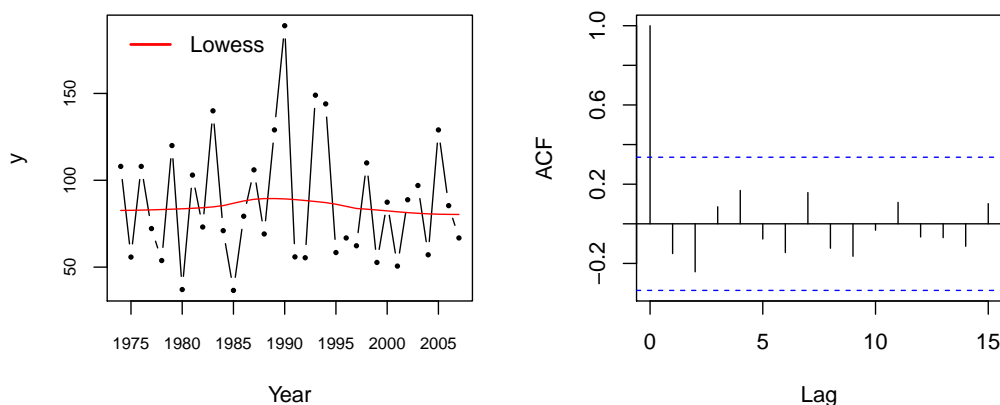


Figure 4: (a) Time series and (b) auto-correlation plot the daily flow (in m^3/s) dataset. For (a), the red smoothed line corresponds to a lowess fit.

Note that, for convenience, the data series was copied into a new variable named `y`. More information about that dataset can be obtained by running `help(station01AJ010)`. Figure 4

shows a time series and an auto-correlation plot for the new dataset. One of the most important things when modeling this type of data using some univariate probability distribution is to first make sure that the observations do in fact approximately behave as independent realized values of some common random variable. In other words, the observations must not only be approximately independent; it must also be reasonable to assume that they were all generated by the same process. Although those assumptions can rarely be fully verified in practice, they can still reasonably be made for many examples. In this case, Figure 4 does not suggest any time trends or serial dependence, such that it seems reasonable to model that dataset using a standard univariate distribution. Coles (2001) dedicates a chapter to modeling of extreme data characterized by time trends, but that topic will not be addressed here, as it cannot be tackled using `FAMle`.

Since the goal here is to demonstrate how `boot.mle` may be used, and that the latter requires a `mle` object, the first thing to do is to fit a distribution to `y`. Without justifications, the gamma distribution will temporarily be fitted to `y` for illustration's sake, although an arguably more appropriate model will be entertained for this same dataset in Section 6.

```
> fit.y <- mle(x=y,dist='gamma',start=c(.1,.1))
> fit.y
```

```
-----
                Maximum Likelihood Estimates
-----
Data object:  y
Distribution:  gamma

----- Parameter estimates -----

      shape.hat  rate.hat
Estimate    6.437  0.07375
Std.err     1.520  0.01811

----- Goodness-of-Fit -----

  log.like      aic      ad      rho
-166.6992  337.3983   0.4172  0.9889
-----
```

```
> plot(x=fit.y,ci=TRUE,alpha=alpha)
```

Judging by Figure 5, the gamma distribution does seem to be fitting quite well in this case. Nonetheless, of greater interest here are the parameter estimates and the associated estimators' approximate standard errors, as shown by the previous code chunk. As well explained in `help(mle)`, the parameter estimators' standard errors are computed/approximated here using the observed Fisher's information matrix, i.e., the negative of the inverse of the Hessian matrix evaluated at the estimated parameter values (note: here since `optim` is used to minimize the negative of the log-likelihood function, the Hessian is taken as returned and inverted to yield the observed Fisher's information). Under certain conditions (not discussed here), the maximum likelihood parameter estimates may be regarded as realized

Diagnostic plots : $y \sim \text{gamma}$

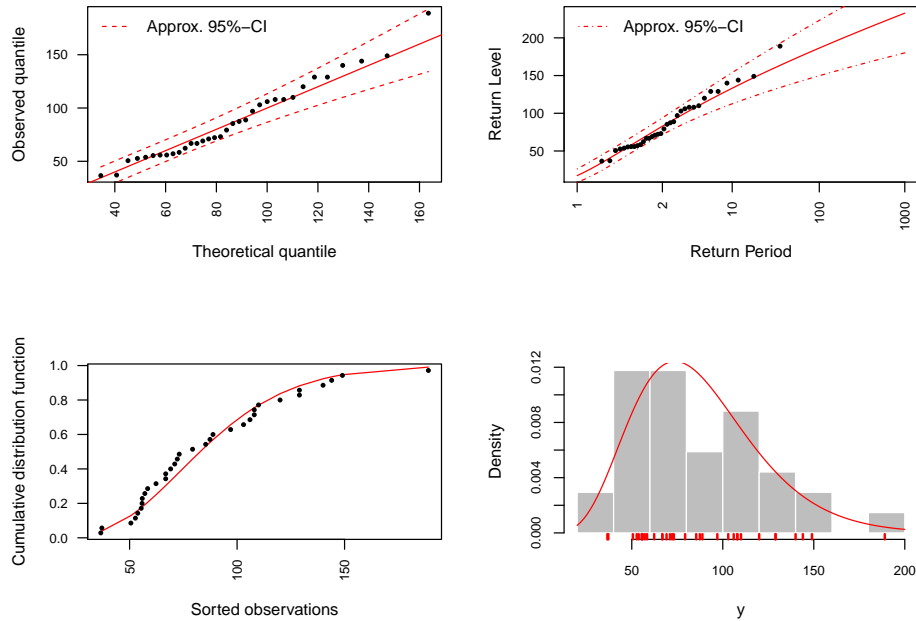


Figure 5: Diagnostic plots for the gamma model fitted to the `station01AJ010` dataset. The dashed red lines correspond to the lower and upper confidence bounds of the approximate 95% confidence intervals derived using the observed Fisher's information matrix in conjunction with the well-known delta method.

values of random variables (their estimators) that are approximately distributed according to a multivariate normal distribution with mean corresponding to the true parameter values and covariance matrix approximately equal to Fisher's information evaluated at the true parameter values. Obviously, the true parameter values are almost never known, such that the covariance matrix has to be estimated by either plugging the fitted parameter values into Fisher's information, or by simply using the observed Fisher's information. In this case, the second option is being use for generality. All of this is done automatically by `mle`, but, for clarity, the following code chunk shows how this may be done by directly using the optimization output provided by `optim` – see `help(mle)` and `help(optim)`:

```
> names(fit.y)

[1] "fit"      "x.info"   "dist"     "par.hat"  "cov.hat"  "k"
[7] "n"       "log.like" "aic"      "ad"       "data.name" "rho"

> cov.hat.y <- solve(fit.y$fit$hessian)
> se.asy <- sqrt(diag(cov.hat.y))
> se.asy

[1] 1.52018615 0.01810993
```

Now, it is well known that those standard errors correspond to large sample approximations and that they may be inaccurate for relatively small sample sizes. In this case, only 34 observations are available for y , and it might be suspected that the large sample approximations motivated by maximum likelihood theory may fail. When this happens, it is usually deemed safer to assess the parameter estimators (joint) sampling distribution by Monte Carlo methods. In other words, it is quite easy, given a fitted model, to run a simulation experiment in order to assess the variability of some quantities of interest in light of some theoretical model. Apart from being computationally intensive, the task at hand is relatively simple. In fact, the `boot.mle` function was written to carry out that task, and the following code chunk illustrates how the latter function may be used to obtain bootstrap samples from the fitted model `fit.y`.

```
> boot.y <- boot.mle(model=fit.y,B=2500)
> names(boot.y)

[1] "model"          "B"              "seed"           "par.star"       "gof"
[6] "p.value"        "failure.rate"   "total.time"

> se.boot <- apply(boot.y[['par.star']],2,sd)
> cbind(asymptotic.se=se.asy,bootstrap.se=as.numeric(se.boot))

      asymptotic.se bootstrap.se
[1,]    1.52018615    1.83483341
[2,]    0.01810993    0.02174935
```

In addition to illustrating how bootstrap samples may be generated using `boot.mle`, the above code chunk shows how the latter's output may be used for computing the parameter estimators' standard errors. These values are then compared with those previously obtained using the observed Fisher's information. Clearly, the two do not tell the same story: one of the the bootstrap standard errors is substantially larger than its large sample counterpart. Figure 6, generated from the code chunk shown below, further illustrates that difference.

```
> z1 <- seq(-4*se.asy[1],4*se.asy[1],length.out=100)+fit.y[['par.hat']][1]
> z2 <- seq(-4*se.asy[2],4*se.asy[2],length.out=100)+fit.y[['par.hat']][2]
> fz12 <- outer(z1,z2,
+   function(h1,h2) dmvnorm(cbind(h1,h2),fit.y[['par.hat']],fit.y[['cov.hat']])
+ )
> contour(z1,z2,fz12)
> title(xlab=colnames(boot.y[['par.star']])[1])
> title(ylab=colnames(boot.y[['par.star']])[2])
> points(boot.y[['par.star']],cex=.1,col='red',pch=19)
```

The `boot.mle` function returns other arguments that are not discussed here, but some of them will be discussed in the next section – see `help(boot.mle)` for more information.

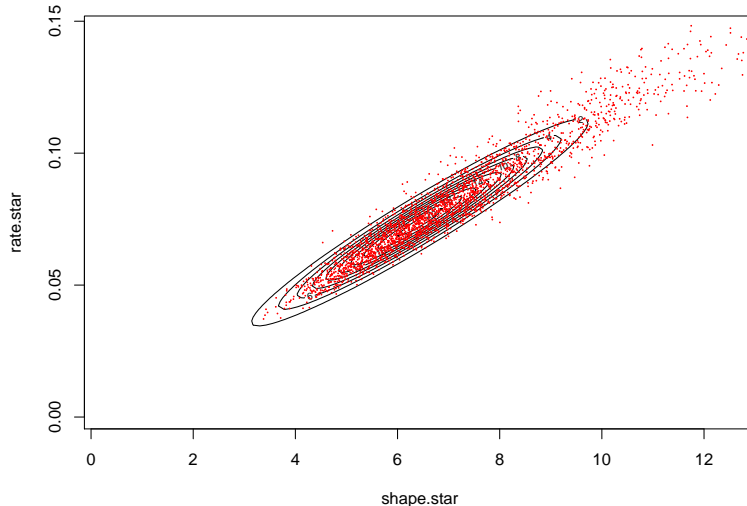


Figure 6: Scatter plot of the realized values drawn from the parametric bootstrap distribution of the maximum likelihood parameter estimates obtained from `fit.y`. The contour lines illustrate a bivariate normal density with covariance matrix equal to the observed Fisher's information and with mean taken to be the parameter estimates contained in `fit.y`. In itself, that bivariate normal density has no particular meaning; it is only shown here as a means of illustrating the fact that the bootstrap distribution of the gamma shape parameter yields a larger variance than does its large sample approximation counterpart.

5 Approximate Confidence Intervals and Hypotheses Tests

This section deals with two main topics: quantile confidence intervals and composite hypotheses tests. As stated in Section 1, different types of applications will require different types of inferences. Although `FAMle` may be useful for carrying tasks in several fields of application, only examples with hydrometeorological variables will be considered for the remaining part of this document. For that type of applications, for example, one is often concerned with making inferences regarding theoretical quantiles corresponding to extreme meteorological events that occur with small probabilities. But before discussing quantile confidence intervals, the concept of goodness-of-fit testing is briefly addressed.

As already mentioned in Section 3, `mle` computes and returns goodness-of-fit statistics. Although, for a particular example, those values may be used as a means of choosing between several fitted distributions, it is also possible to use some of the latter in order to construct simple hypotheses tests. For example, `ad` is known to give more weight to observations in the tails of the distributions than to those in the center, such that is often regarded as a good statistic for detecting departures from distributional assumptions in the tails. As another example, Pearson's correlation coefficient between the fitted quantiles and the observed ones may also be useful for detecting poor fits. That said, a natural question to ask when fitting a parametric model to a given dataset is the following: "*Does it make sense to assume that the*

observed data could actually have been generated from the fitted model”? To put it another way, to what extent is the fitted model consistent with the data that was used in fitting it? An answer to that question may be obtained by constructing a simple hypothesis test based on some observed statistic that is particular to either or both the observed data and the fitted model. For example, if `ad` was used to assess the fitted model’s plausibility in light of the data, the goal would be to check at what level the observed `ad` statistic fell into its respective sampling distribution. But since, in practice, the true parameter values are seldom known, the true sampling distribution for the statistic of interest cannot be obtained (unless, of course, that the statistic is the same for all possible parameter values). An alternative is thus to derive a test that is conditional on the fitted model. In other words, if the data has in fact been generated from the fitted model, then what is the probability of observing a value as extreme or more extreme than the statistic that was actually observed. Here, for the `ad` statistic, the goal would be to compute the probability of observing a value as high or even higher than what was observed – this is the definition of the well-known *p-value*.

That said, given the fitted gamma model obtained for the previous example, the idea is to draw many samples from the fitted model, and for each new sample, to re-estimate the parameters and to compute a new realized `ad` statistic. That way, the hypothetical sampling distribution of `ad` will be empirically approximated and then be used to approximate the p-value of the test. It turns out that this can be achieved using `boot.mle`. In fact, `boot.mle` does return the p-values for the two tests discussed here:

```
> boot.y[['p.value']]

      ad      rho
0.3454618 0.5985606
```

Thus, based on these two criteria, since observing values as extreme or more extreme than the latter two could likely be the result of chance alone, there are no reasons why one should doubt the hypothesis according to which the gamma distribution is a plausible model for the data at hand. To put it another way, the model is consistent with the data. For more information regarding composite hypotheses tests as well as so-called parametric bootstrap tests, the interested reader is referred to the following two excellent books: Cox & Hinkley (1979) and Davison & Hinkley (1997). Finally, it should be recalled that tests such as those derived above are often considered as highly hypothetical (e.g., Rice, 2003; Cox & Hinkley, 1979), as no such thing as a true distribution does exist for most practical situations. Nonetheless, in many instances, those tests are still very useful for quickly assessing the fitted model’s plausibility.

So far, from both Figure 5 and the tests results shown above, it seems reasonable to look at the fitted gamma distribution as a suitable model for the purpose of describing the `station01AJ010` dataset, such that it would now seem appropriate to move on to statistical inference for some quantities of interests, in this case, the quantile estimators for some high return periods. More precisely, for a return period T , Q_T corresponds to the quantile value which is expected, in a series of independent samples of equal size n , to occur (or to be exceeded) no more than once every T -time period – for the random variable X with cumulative distribution function F , $Q_T = F^{-1}(p)$ if $p = 1 - 1/T$. In this case, the return period T is in years. That said, for the `station01AJ010` dataset, one may wish to obtain quantile estimates for the following return periods:

```

> RP <- c(2,10,20,50,100,500)
> p <- 1 - 1/RP
> p

[1] 0.500 0.900 0.950 0.980 0.990 0.998

```

Using the fitted model stored in the variable `fit.y`, the `distr` function may be used to obtain the required values; i.e.,

```

> Q.hat <- distr(x=p,model=fit.y,type='q')
> Q.hat

[1] 82.81087 133.25210 150.48042 171.49443 186.47899 219.25191

```

Although those values are interesting, it is often more informative to get an idea about the potential variability that is to be expected when estimating them. For example, it is often convenient to summarize frequency analyses in terms of confidence intervals for the quantiles of interest. There exist numerous ways in which approximate confidence intervals can be derived for any particular problem. Here, for the example's sake, the following four types of intervals are entertained:

- Approximate normal quantile confidence interval derived using the observed Fisher's information matrix in conjunction with the delta method – see, e.g., Rice (2003) and Coles (2001) regarding the delta method;
- The same interval as the previous, except that the computations are carried out on the natural logarithmic scale and later exponentiated to yield the interval on the original scale;
- Approximate bootstrap quantile confidence interval obtained using the so-called *percentile* method (see, e.g., Davison & Hinkley, 1997);
- And the so-called *basic bootstrap interval* (method also often referred to as *reflexion method*) (see, e.g., Davison & Hinkley, 1997).

For a given numerical example, the first two intervals can be obtained using the `Q.conf.int` function, while the second two can be obtained from `Q.boot.ci` (the reader should now consult `help(Q.conf.int)` and `help(Q.boot.ci)`). While the former function requires a `mle` object as main argument, the latter requires an object returned by `boot.mle`. The following code chunk illustrates how all four intervals may be obtained for the previously specified return periods (in this case, `alpha` equals 0.05, such that 95% confidence intervals are obtained):

```

> Q.conf.int(p=p,model=fit.y,alpha=alpha,ln=FALSE)[c(1,3),]

      p = 0.5   p = 0.9   p = 0.95   p = 0.98   p = 0.99   p = 0.998
low-2.5% 71.64955 112.5527 125.0045 139.6325 149.7839 171.3921
up-97.5% 93.97219 153.9515 175.9563 203.3564 223.1741 267.1117

```



```

> Q.conf.int(p=p,model=fit.y,alpha=alpha,ln=TRUE)[c(1,3),]

      p = 0.5  p = 0.9  p = 0.95  p = 0.98  p = 0.99  p = 0.998
low-2.5% 72.36903 114.0803 127.0444 142.4173 153.1687 176.2556
up-97.5% 94.75932 155.6458 178.2398 206.5082 227.0333 272.7369

> Q.boot.ci(p=p,boot=boot.y,alpha=alpha)[['percentile']][c(1,3),]

      p = 0.5  p = 0.9  p = 0.95  p = 0.98  p = 0.99  p = 0.998
low-2.5% 72.45505 112.1003 124.9385 139.6009 150.0482 171.7810
up-97.5% 93.95050 153.7670 176.0941 204.0852 224.4526 269.1219

> Q.boot.ci(p=p,boot=boot.y,alpha=alpha)[['reflexion']][c(1,3),]

      p = 0.5  p = 0.9  p = 0.95  p = 0.98  p = 0.99  p = 0.998
low-2.5% 71.67124 112.7372 124.8667 138.9037 148.5053 169.3819
up-97.5% 93.16669 154.4039 176.0223 203.3880 222.9098 266.7228

```

Before commenting on those confidence intervals, it is informative to look at Figure 7 for a visual assessment of the estimated quantiles' bootstrap distributions. The following code chunk illustrates how those distributions may be obtained and plotted. Also visible from that code is the `delta.Q` function, which is used to obtain the quantile estimates and their estimators' standard errors.

```

> Q.boot.dist <- sapply(as.list(p),function(h) distr(h,dist=fit.y[['dist']],
+ param=boot.y[['par.star']],type='q'))
> Q.norm <- sapply(as.list(p),function(h) delta.Q(p=h,model=fit.y,ln=FALSE))
> layout(matrix(1:6,nc=2))
> for(i in 1:ncol(Q.boot.dist)) {
+ hist(Q.boot.dist[,i],freq=F,col='gray',border='white',
+ main=paste('T = ',RP,sep='')[i],xlab='',cex.axis=.75,las=2)
+ curve(distr(x=x,dist='norm',param=Q.norm[,i],type='d'),
+ add=TRUE,col='red')
+ }

```

From Figure 7, it can be observed that the large sample quantile variance approximations are somewhat consistent with the bootstrap variance counterparts, regardless of what was previously observed for the parameters themselves (see Figure 6). In other words, although the large sample shape parameter variance approximation was clearly less than that of its bootstrap counterpart, it does not seem to be the case for the quantile variance estimates discussed here. In addition, most of the quantile bootstrap distributions are nearly symmetrical, which might explain why the different types of confidence intervals shown above yield results that are somewhat similar (except for the normal intervals computed on the log-quantile scale)

Finally, there are a two facts that must be recalled about the approximate quantile confidence intervals presented and discussed in this section:

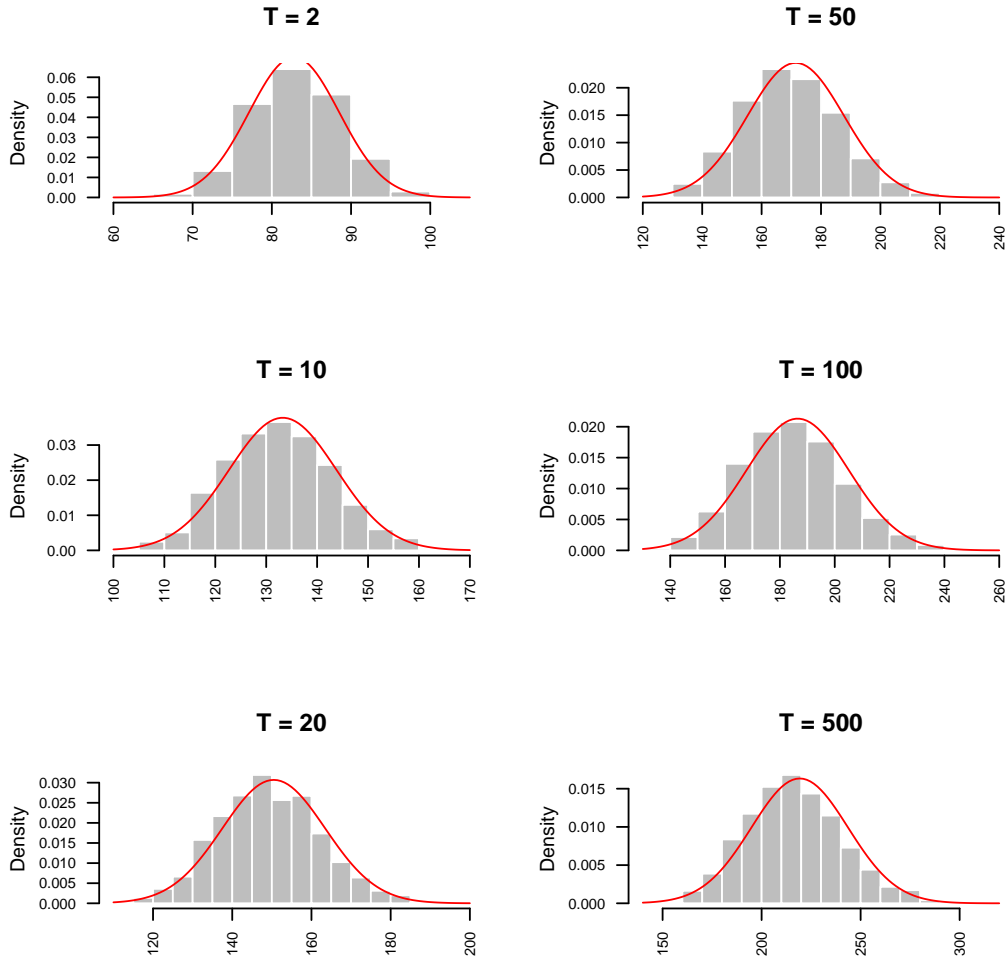


Figure 7: Histograms for the quantile bootstrap distributions with return periods $T = \{2, 10, 20, 50, 100, 500\}$, which were obtained from the fitted gamma to the `station01AJ010` dataset. The red curves – normal densities with scale parameter equal to the approximate quantile variance and location parameters taken to be equal to the respective quantile estimates – are added to illustrate the similarities between the sampling variance estimated using the normal approximation (in conjunction with the delta method) and the actual spread of the bootstrap distributions. In this case, it appears that the bootstrap distributions are somewhat centered around the maximum likelihood quantile estimates.

- The numerical intervals presented in the application above are not random – they are realized values of some random intervals. This means that those realized intervals either contain or do not contain the true quantile value that is being estimated. That said, when interpreting such intervals, a probability statement is not being made with regard to the true quantile value, but rather with regard to the statistical procedure that is used to estimate it. In other words, if the fitted model is indeed the true distribution from which the observed data have arisen, and that the experiment (i.e., data collection, estimation, and confidence intervals derivation) could be repeated many times, then, for a nominal level $1 - \alpha$, the long run average of the intervals that do succeed in covering the true quantile value would lie close to $1 - \alpha$ (it should be noted, however, that poor coverage probabilities might still occur depending on the type of interval used). Obviously, as it was said for the hypotheses tests derived previously, the present definition is also quite hypothetical. Nonetheless, it is still very informative with respect to the variability that should be expected when estimating theoretical quantiles using parametric distributions.
- The bootstrap intervals and the normal intervals that are discussed in this document are not exact – they are approximate confidence intervals. Roughly, for the bootstrap intervals, it is assumed that the true maximum likelihood quantile estimator’s sampling distribution – sampling distribution that would be obtained from generating random samples from the true parametric model – can be fairly well approximated by the hypothetical sampling distribution obtained from bootstrapping the estimated model. On the other hand, roughly, the normal intervals assume that the maximum likelihood quantile estimate corresponds to a realized value of a random variable (its estimator) that is approximately normally distributed with mean equal to the true quantile value and variance that may be fairly well approximated using Fisher’s information matrix evaluated at the true parameter values (in conjunction with the delta method). Here, however, since the true parameter values are not known, Fisher’s information may be approximated by either the observed Fisher’s information (as it is the case here) or by evaluating Fisher’s information at the estimated parameter values. Obviously, this second approach assumes more than the bootstrap (see Davison & Hinkley (1997) and Rice (2003) for good discussions about confidence intervals).

6 Declaration of New Probability Distributions

In the previous section, a numerical example in which a gamma model was fitted to a dataset of annual maximum average daily flows (in m^3/s) was considered. At that point, it was assumed, based on both visual assessments and goodness-of-fit testing, that the latter family of distributions was suitable for the purpose of describing the dataset at hand. Nonetheless, it was briefly mentioned that other families of distributions were sometimes to be preferred, based on certain theoretical arguments, when dealing with such extreme data. More precisely, when dealing with, for example, annual maximum average daily flows, the so-called *generalized extreme value* (GEV) distribution is often regarded as a plausible model – under certain conditions (e.g., Coles, 2001), extreme observations will likely start to behave as realized values of a random variable asymptotically distributed (this is the concept of convergence in distribution) according to one of three types of limiting distributions; GEV containing each of them three as special cases.

Since the `station01AJ010` dataset is consistent with the extreme value framework described

by Coles (2001), it would be interesting to fit GEV to that dataset. However, functions for the GEV model are not available in R. Although there exist numerous packages on CRAN which do contain those functions, the latter are relatively easy to program, and this is what will be done in this section. In fact, since they will be written with the intention of later being used with the `FAmle` functions, it will be important to make sure that those new functions are consistent with all such functions already available in R. That way, no problems should occur when they are used to carry out tasks similar to those that have so far been discussed in this tutorial.

```
> dGEV <- function(x,shape=1,scale=1,location=0,log=FALSE)
+ {
+   fx <- 1/scale*(1+shape*((x-location)/scale))^(1/shape-1)*
+     exp(-(1+shape*((x-location)/scale))^(1/shape))
+   if(log) return(log(fx))
+   else return(fx)
+ }
> pGEV <- function(q,shape=1,scale=1,location=0,lower.tail=TRUE,log.p=FALSE)
+ {
+   Fx <- exp(-(1+shape*((q-location)/scale))^(1/shape))
+   if(!lower.tail) Fx <- 1 - Fx
+   if(log.p) Fx <- log(Fx)
+   return(Fx)
+ }
> qGEV <- function(p,shape=1,scale=1,location=0,lower.tail=TRUE,log.p=FALSE)
+ {
+   if(log.p) p <- exp(p)
+   if(!lower.tail) p <- 1 - p
+   xF <- location+scale/shape*((-log(p))^(shape)-1)
+   return(xF)
+ }
> rGEV <- function(n,shape=1,scale=1,location=0)
+   qgev(runif(n),shape,scale,location)
```

Now that the GEV distribution has been added into the R environment, it can be fitted, for example, to `y`, and the obtained `mle` object can be used as usual:

```
> fit.y.gev <- mle(x=y,dist='GEV',c(.1,1,1))
> fit.y.gev
```

```
-----
                Maximum Likelihood Estimates
-----
Data object:  y
Distribution:  GEV

----- Parameter estimates -----
                shape.hat scale.hat location.hat
Estimate    0.08264    26.175    69.866
```

```
Std.err      0.17032      4.076      5.291
```

```
----- Goodness-of-Fit -----
```

```
log.like      aic      ad      rho
-166.4371  338.8741    0.3552    0.9923
-----
```

Finally, the code chunk shown below is presented as a means of illustrating how inferences might greatly differ when going from one plausible model to another; in this case, the previously fitted gamma model vs. the new GEV model.

```
> Q.conf.int(p=p,model=fit.y,alpha=alpha)
```

```
      p = 0.5  p = 0.9  p = 0.95  p = 0.98  p = 0.99  p = 0.998
low-2.5% 71.64955 112.5527 125.0045 139.6325 149.7839 171.3921
Estimate 82.81087 133.2521 150.4804 171.4944 186.4790 219.2519
up-97.5% 93.97219 153.9515 175.9563 203.3564 223.1741 267.1117
```

```
> Q.conf.int(p=p,model=fit.y.gev,alpha=alpha)
```

```
      p = 0.5  p = 0.9  p = 0.95  p = 0.98  p = 0.99  p = 0.998
low-2.5% 67.68637 106.9238 114.1674 114.1445 106.5823 60.69746
Estimate 79.60677 134.6031 157.9841 190.3891 216.3609 282.43311
up-97.5% 91.52718 162.2824 201.8008 266.6336 326.1396 504.16876
```

As visible from the previous output, quantile uncertainty is much more pronounced for the GEV model, but this is to be expected since it has one more parameter than gamma. Moreover, given that only 34 observations were used in fitting those two models, half of the quantile estimates correspond to extrapolations (i.e., those for the return periods $T = \{50, 100, 500\}$), such that it is not unreasonable at all to observe large uncertainties at those levels (again, this is especially true for GEV).

7 The R package FAdist

The previous section has dealt with the declaration of functions that pertain to a new univariate distribution that is not available in the R environment; i.e., the GEV distribution. In this section, a package containing such functions for various distributions is presented. Actually, not much needs to be said here; the only thing being that the package name is `FAdist`¹, and that, at the time of writing of this document, the latter could be downloaded from CRAN. Once downloaded, the `FAdist` package may be installed and loaded (see `help(package='FAdist')` for more information).

Just to give a quick example, in the code chunk below, a three-parameter lognormal distribution, made available when loading `FAdist`, is fitted to `y` (note that this new distribution is based on the R function `lnorm`):

¹See Aucoin (2011).

```
> library(FAdist)
> fit.y.ln3 <- mle(x=y,dist='lnorm3',start=c(1,1,1))
```

8 Bayesian Model Estimation Using metropolis

All of the functions and examples that have so far been presented and discussed in this document rely upon maximum likelihood as a means of estimating the unknown parameters, and upon a frequentist framework as a means of carrying out statistical inferences. This section is different; it deals with a Bayesian approach to parameter estimation and statistical inference. Since the goal here is to present the main functions and to keep the discussion as brief as possible, the interested reader is referred to the classical book by Box & Tiao (1973) for an interesting discussion on the different types of statistical inferences, and to the excellent book by Gelman *et al.* (2004) for a modern treatment of Bayesian methods in general.

It should also be said here that the `FAmle` package was originally designed as a tool for carrying out tasks pertaining to univariate frequency analysis within a maximum-likelihood-based frequentist framework. Nonetheless, the `metropolis` function discussed in this section was later added to the package, as it seemed natural to do so given that it mainly relies upon `distr` and `mle`. As it is the case for many of functions available in `FAmle`, the `metropolis` function was written in order to be as general as possible.

The `metropolis` function is based on a specific version of the Metropolis algorithm (Metropolis *et al.*, 1953), such that it approximates the posterior distribution using a Markov Chain - the returned distribution corresponds to time correlated draws from the posterior distribution (see the details in `help(metropolis)`). The algorithm used is well described by Gilks *et al.* (1996), Carlin & Louis (2009), and Gamerman & Lopes (2006). Here, no attention will be given to topics such as tuning of the Metropolis algorithm or MCMC convergence; only examples on how to use the `metropolis` function will be presented. For more information about those topics, the reader is again referred to excellent texts such as Gilks *et al.* (1996), Carlin & Louis (2009), Gelman *et al.* (2004), and Gamerman & Lopes (2006), as well as to the various references given therein. Finally, before continuing with the examples below, the reader should carefully examine the help file available for `metropolis`.

The first dataset that will be used in this section is taken from Coles (2001) and consists of 64 sea level (in meters) yearly maxima for the time period 1923-1987 – here it will be referred to as the `ColesData` dataset (see `help(ColesData)` for more detail).

```
> data(ColesData)
> z <- ColesData[,2]
```

As suggested by Coles (2001), the GEV distribution is a plausible model for that dataset. That said, the code chunk below shows how the GEV distribution is first fitted to that dataset using `mle` (see Figure 8).

```
> fit.z <- mle(x=z,dist='gev',start=c(.1,1,1))
> plot(x=fit.z,ci=TRUE,alpha=alpha)
```

As visible from the previous output, for the `dist` argument, `gev` was specified instead of the

Diagnostic plots : $z \sim \text{gev}$

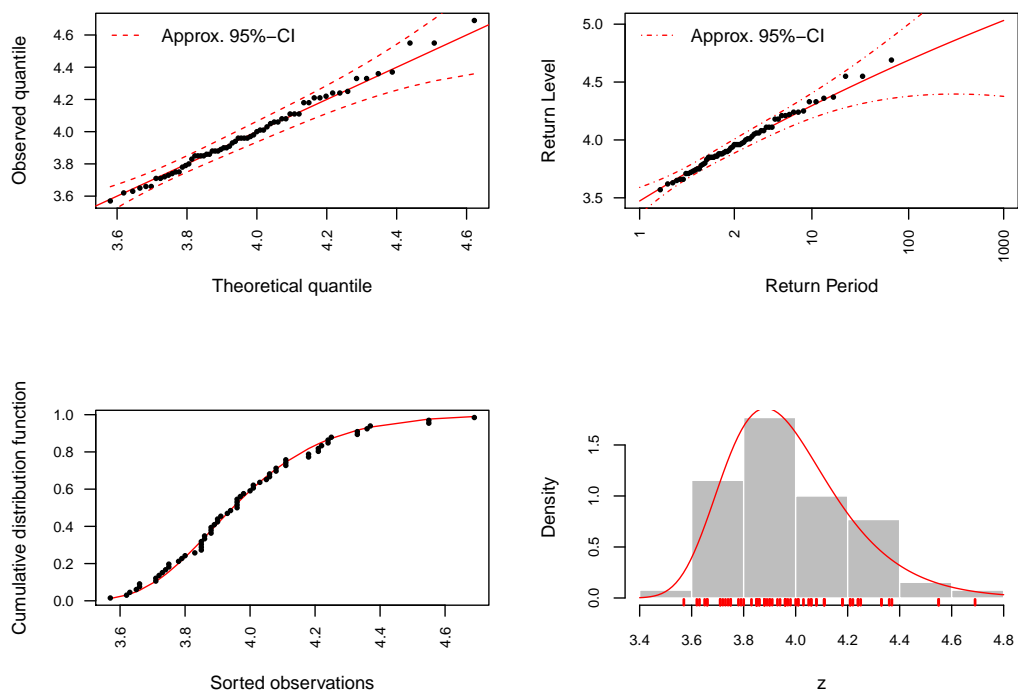


Figure 8: Diagnostic plots for the gev model fitted to the `ColesData` dataset. The dashed red lines correspond to the lower and upper confidence bounds of the approximate 95% confidence intervals derived using the observed Fisher's information matrix in conjunction with the well-known delta method.

previously declared `GEV`. This is because the `gev` functions were made available into R when loading the `FAdist` package at the previous section.

A `mle` object having been obtained for `z`, the code chunk below shows how it may be provided to `metropolis` in order to carry out a full Bayesian estimation of the `gev` parameters (see Figure 9).

```
> trans.z <- list(function(x) x, function(x) exp(x), function(x) x)
> bayes.z <- metropolis(model=fit.z, iter=10000, tun=2, trans.list=trans.z)
> plot(x=bayes.z, plot.type='carlin')
> bayes.z
```

```
-----
                    Bayesian Posterior Summary
-----
Data object:  z
Likelihood:  gev
Prior distribution specified: No
-----
Marginal posterior distributions
---
      shape  scale  location
mean -0.0259 0.2033  3.8719
sd    0.1005 0.0203  0.0282
1%   -0.2232 0.1614  3.8060
50%  -0.0333 0.2016  3.8711
99%   0.2421 0.2573  3.9379
-----
Posterior correlations
---
      shape  scale  location
shape    1.00 -0.32  -0.40
scale   -0.32  1.00   0.32
location -0.40  0.32   1.00
-----
Number of iterations: 10000
Burnin: first 0 iterations
Acceptance rate: 0.3034
Required time: 0.053 minutes
-----
```

It is very important that the argument `trans.list` be fully understood, as it is the most important argument to be provided to `metropolis`! As explained in the help file provided for `metropolis`, the version of the Metropolis algorithm that is implemented here requires, at least as much as possible, that all parameters be made as **normal** as possible. In other words, in order to yield a good approximation, it is important (but not mandatory) that all parameters be defined on the real line before implementing the algorithm. For the `GEV` distribution, as it is parametrized here, both the shape and location parameters are defined on the real line, but the scale parameter can only take on positive values. That said, it needs

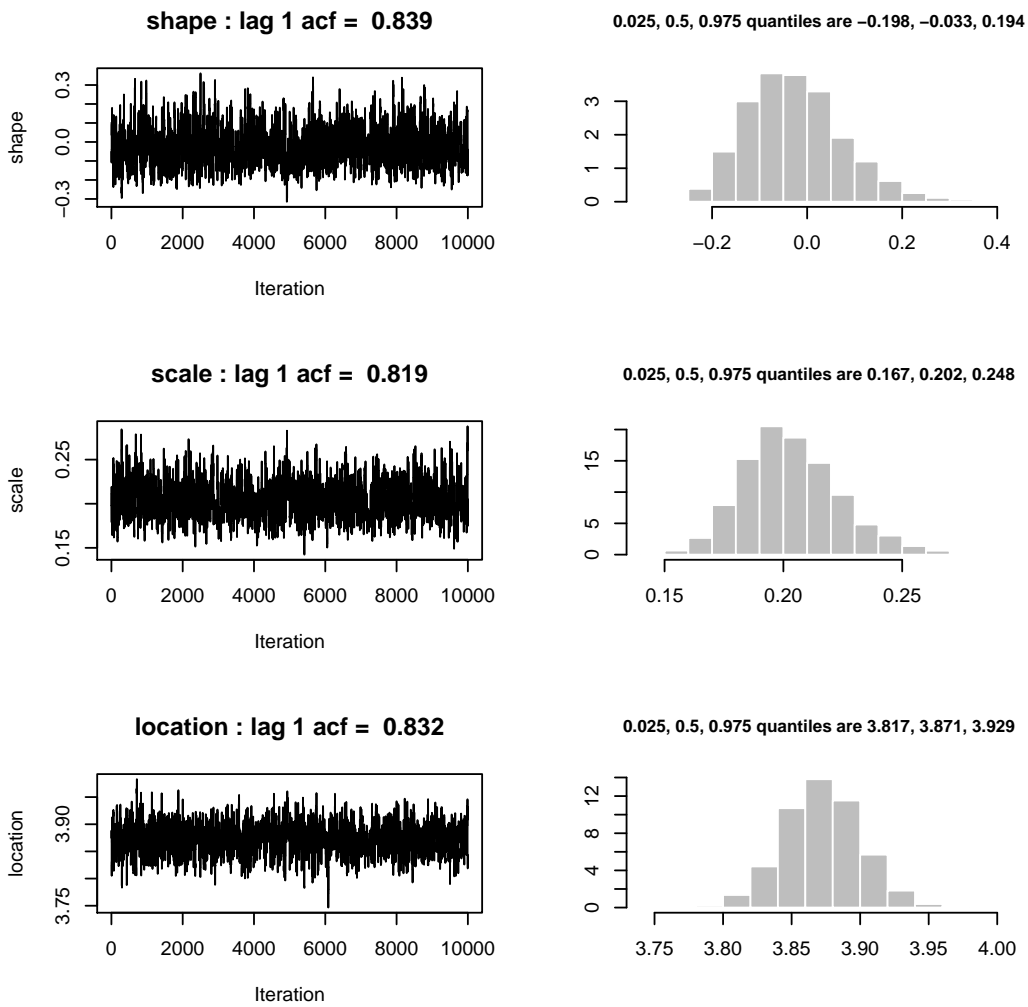


Figure 9: Marginal trace plots and histograms for the parameters of the GEV model fitted to the ColesData dataset.

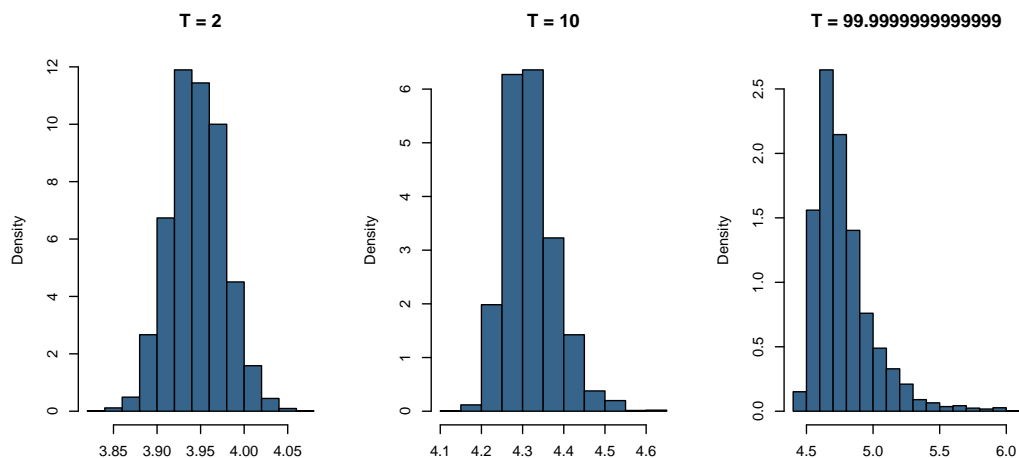


Figure 10: Quantile posterior distributions obtained using the `bayes.z` object.

to be transformed before implementing the algorithm. Here, a natural logarithmic transformation is used, whose inverse transformation corresponds to the well-known exponential function. That said, the `trans.list` requires a list containing the inverse transformations in order to carry out the simulation on the correct parameter space. If that argument is left unspecified, the function will assume that all parameters are indeed defined on the real line, and will perform the simulation accordingly. However, it should be noted that the `metropolis` function will return the marginal posterior distributions on their original scales. Finally, if the `prior` argument is left unspecified, as it was the case in the previous example, the function will assume that the prior is proportional to 1 on the parameter space defined by `trans.list` – a prior distribution proportional to 1 is called improper, as it is not a density.

Once draws from the posterior distribution have been obtained, they can be used to compute the posterior distribution for any quantity that is a function of the estimated parameters. For instance, interest here might be in computing the posterior distributions for some specific quantiles, as illustrated by the code chunk below and displayed by Figure 10.

```
> p <- c(.5, .9, .99)
> Q.p.post <- sapply(as.list(p),
+   function(h) distr(x=h, dist=fit.z[['dist']],
+     param=bayes.z[['sims']], type='q'))
> layout(matrix(1:length(p), nr=1))
> for(i in 1:ncol(Q.p.post)) hist(Q.p.post[,i],
+   freq=FALSE, col='steelblue4',
+   main=paste('T = ', 1/(1-p[i]), sep=''), xlab='')
```

Depending on the situation at hand, it is sometimes deemed safer to consider a non-informative (or only weakly informative) prior distribution that is nonetheless proper. This will insure that the resulting posterior is also proper. Although this might not be necessary here, specification of the prior distribution will allow to illustrate how this is done with `metropolis`. Again, it should always be kept in mind that the prior distribution will be

used on the transformed parameter space, such that it must be defined accordingly. Here's an example:

```
> prior.z.weak <- function(x)
+   dnorm(x[1],0,1000)*dnorm(x[2],0,1000)*dnorm(x[3],0,1000)
> bayes.z.weak <- metropolis(model=fit.z,iter=10000,
+   tun=2,trans.list=trans.z,prior=prior.z.weak)
```

The following code chunk illustrates an equivalent way of specifying the previous prior distribution, which suggests that fairly complex prior distributions may be specified using `metropolis`.

```
> prior.z.weak.2 <- function(x) dmvnorm(x=x[1:3],
+   mean=rep(0,3),sigma=diag(1000,3))
> bayes.z.weak.2 <- metropolis(model=fit.z,iter=10000,
+   tun=2,trans.list=trans.z,prior=prior.z.weak.2)
```

To help further illustrate the point made at the previous paragraph, a final example is presented; an example which corresponds to an application in hydrology, for which the main goal is to enhance flood estimates for large return periods at hydrometric stations where very few historical data are available. In this last example, the goal will therefore be to make use of flood information available at the regional scale in order to elicit a prior distribution for the frequency distribution parameters at a particular site of interest.

The first thing to do is to load the dataset into R:

```
> data(floodsNB)
> length(floodsNB)

[1] 64

> names(floodsNB)[1:5]

[1] "01AD002" "01AD003" "01AD004" "01AE001" "01AF009"

> floodsNB[[3]]

$data
1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979
2660 3710 2710 2740 2700 3740 3480 2290 2660 2790 2780 4200

$peak
1968 1969 1970 1972 1973 1974 1975 1978 1979
2690 3790 2810 2730 3790 3600 2320 2820 4250

$l.n.drain
[1] 9.648595
```

```

$coor
  Latitude Longitude
2 47.36078 -68.32489

$status
[1] "Active"

$Aucoin.2011
[1] TRUE

```

At large, `floodsNB` is a list whose elements correspond to individual hydrometric stations located across the Canadian province of New Brunswick (NB), and, in this case, the variable of interest is `peak` and corresponds to *annual maximum daily peak discharge observations* (in m^3/s). As visible from the previous code chunk, the dataset contains flood data from 64 stations located across NB. However, not all stations will be analyzed here – the stations considered for analysis are identified in `Aucoin.2011` in the list; i.e.,

```

> aucoin.2011 <- names(which(sapply(floodsNB,
+   function(i) i[['Aucoin.2011']]))))
> aucoin.2011

[1] "01AD002" "01AD003" "01AD004" "01AE001" "01AF009" "01AK001" "01AK006"
[8] "01AK007" "01AL004" "01AM001" "01AN002" "01AP002" "01AP006" "01AQ001"
[15] "01BE001" "01BJ003" "01BJ007" "01BJ012" "01BL002" "01BL003" "01B0001"
[22] "01BP002" "01BU002" "01BU003" "01BV004" "01BV006" "01AJ007" "01AK008"
[29] "01AL003" "01AL005" "01AL006" "01AL007" "01AL008" "01AN001" "01A0009"
[36] "01AP009" "01AQ008" "01AR008" "01BJ001" "01BL001" "01BR001" "01BU004"

> length(aucoin.2011)

[1] 42

> NB <- list()
> for(i in names(floodsNB))
+   if(is.element(i, aucoin.2011)) NB[[i]] <- floodsNB[[i]]

```

Now, before going any further, it should be mentioned that three basic assumptions are made with regards to the annual observations available at the stations retained for analysis:

- The at-site annual observations may be treated as independent realizations of some random variable Y_i , where i denotes a specific station;
- The random variable Y_i arises from a GEV distribution;
- The data generating process at site i is stationary.

The above assumptions will not be explicitly tested here.

For illustration's sake, the station of main interest here is chosen to be 01AJ007, for which 7 peak flood historical observations are available. As already pointed out previously, interest lies in the estimation of quantiles associated with large return periods – for example, the .99th GEV quantile, which corresponds to the 100-year flood. Obviously, estimating the .99th quantile is problematic here, since only 7 observations are available. In fact, given the smallness of the observed flood series, this example will also serve to illustrate how to use `metropolis` without providing an `mle` object to the `model` argument: this feature is very useful since, in some instances, prior information will be available for the distribution's parameters, but not enough data will be available to obtain reliable maximum likelihood estimates (and the corresponding observed Fisher's information matrix) using `mle`. Therefore, without even trying to obtain the maximum likelihood GEV parameter estimates for the data of station 01AJ007, the following code chunk illustrates how to use `metropolis` without resorting to `mle`. Actually, as well explained in `help(metropolis)`, the procedure only requires the specification of a list object containing both the data (`x`) and the hypothesized distribution; in this case

```
> st <- '01AJ007'
> station.x <- NB[[st]]
> ln.drain.x <- station.x[['ln.drain']]
> model.x <- list(x=station.x[['peak']],dist='gev')
> model.x
```

```
$x
 1973 1974 1975 1976 1977 1978 1979
30.00 8.72 5.07 19.10 8.50 6.60 24.40
```

```
$dist
[1] "gev"
```

As already said previously in this document, `metropolis` requires that the distribution's parameters be made as normal as possible through variable transformation. That said, the GEV scale parameter should be transformed so that it is defined on the real line. To this end, a natural logarithmic transformation may be entertained, whose inverse transformation corresponds to the well-known exponential function; i.e.

```
> trans.list.x <- list(function(x) x, function(x) exp(x), function(x) x)
```

The variable `trans.list.x` having been declared, the next step is to elicit a prior distribution for the GEV parameters. For the NB dataset, there exists a well-behaved relationship between site-specific maximum likelihood GEV parameter estimates and their corresponding *drainage area*. Here the drainage area may be extracted as follows:

```
> station.x[['ln.drain']]

[1] 3.443618
```

More precisely, the idea here is to exploit this relationship in order to elicit a prior distribution for the GEV parameters at sites where few historical data are available to yield sensible frequency analyses.

The following code chunk illustrates how the GEV parameters may be estimated by maximum likelihood for the remaining stations.

```
> gev.inits <- function(x) {
+   sig.hat <- sqrt(var(x)*6/pi^2)
+   mu.hat <- mean(x) - digamma(1)*sig.hat
+   return(c(0.01,sig.hat,mu.hat))
+ }
> nb.stations <- nb.stations.fits <- list()
> for(i in names(NB)) {
+   if(substr(i,3,3)=='A' & i != st) {
+     temp <- NULL
+     try(temp <- mle(NB[[i]][['peak']], 'gev', gev.inits(NB[[i]][['peak']])),
+         silent=TRUE)
+     if(!is.null(temp)) {
+       nb.stations[[i]] <- NB[[i]]
+       nb.stations.fits[[i]] <- temp
+     }
+   }
+ }
> f <- function(x) c(x[1],log(x[2:3]))
> nb.stations.par.hat <- t(sapply(nb.stations.fits,
+   function(i) f(i[['par.hat']])))
> colnames(nb.stations.par.hat) <- c('shape', 'ln.scale', 'ln.location')
> nb.stations.par.hat[1:5,]

           shape ln.scale ln.location
01AD002 -0.11022007 6.578624   7.665888
01AD003  0.01527191 4.176540   5.168666
01AE001  0.14614377 3.932903   5.354724
01AF009  0.29278049 2.378894   3.613878
01AK001 -0.05552452 2.433190   3.477394

> nb.stations.ln.drain <- sapply(nb.stations,function(i) i[['ln.drain']])
> nb.stations.regs <- lapply(list('shape', 'ln.scale', 'ln.location'),
+   function(i) lm(nb.stations.par.hat[,i]~nb.stations.ln.drain))
> names(nb.stations.regs) <- colnames(nb.stations.par.hat)
```

The previous output shows several commands, each of which are now explained in detail:

1. `gev.inits` is declared above as a function that generates reasonable initial values, for a given data series, for the optimization algorithm used by `mle` to get the GEV parameter estimates.
2. Information pertaining to stations from sub drainage basin **A** are stored in a new variable called `nb.stations`, where it is also made explicit that station 01AJ007 does

not belong to that group of stations (the data should not be used twice)! At the same time, `mle` is used to estimate the GEV parameters for all stations and the corresponding fitted models are stored in the new variable `nb.stations.fits`. In fact, only stations for which optimization of the likelihood function is successful are retained for prior elicitation – in this context, an optimization failure is often a result of too few historical observations!

3. A function called `f` is declared, which is then used to transform both the GEV scale and location parameters on their natural logarithmic scales.
4. The GEV parameter estimates are extracted from `nb.stations.fits`, transformed and finally stored in a new variable called `nb.stations.par.hat`.
5. For all stations contained in `nb.stations`, the natural logarithms of the corresponding drainage areas are extracted and stored in a variable called `nb.stations.ln.drain`.
6. The natural logarithm of the drainage area is then used as a predictor to fit three independent linear regression models for the previously obtained parameter estimates – see Figure 11

Now, the final step is to make use of the relationships illustrated by Figure 11 in order to elicit a prior distribution for the GEV parameters of station 01AJ007. More precisely, the approach is to propose a multivariate normal prior distribution for the GEV parameters of station 01AJ007, where the mean vector corresponds to the values obtained from the fitted regression models by conditioning on the actual station’s drainage area, and by using the multivariate residuals of these three regression models to estimate a covariance matrix.

```
> mu.x <- sapply(nb.stations.regs,
+ function(i) as.numeric(coef(i)[1]+coef(i)[2]*ln.drain.x))
> mu.x
```

```
      shape    ln.scale ln.location
0.2377576  0.7927161   2.0491200
```

```
> residuals.x <- sapply(nb.stations.regs,residuals)
> sigma.x <- t(residuals.x)%*(residuals.x/nrow(residuals.x))
> sigma.x
```

```
      shape    ln.scale ln.location
shape      0.04378255 -0.03997864 -0.01292101
ln.scale   -0.03997864  0.48904948  0.16824129
ln.location -0.01292101  0.16824129  0.09595529
```

The prior distribution whose parameters are presented in the previous code chunk suggests viewing the extra stations’ GEV maximum likelihood parameter estimates as error-free realized values of some multivariate normal random variable. In other words, it is assumed that those estimates, when considered as a whole, will allow to approximate, with reasonable accuracy, the hierarchical structure that exists between the GEV parameters and the drainage area (as shown by Figure 11). In fact, all stations could be fitted simultaneously

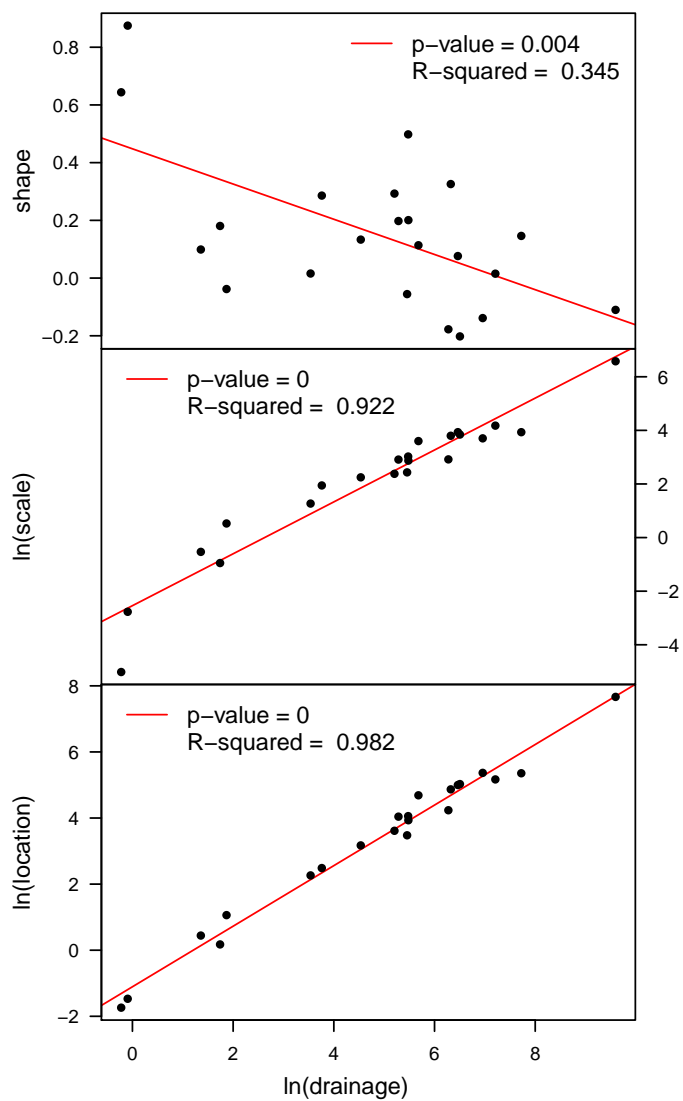


Figure 11: This figure illustrates the relationships between the site-specific maximum likelihood GEV parameter estimates and their corresponding drainage area. Also shown are the regression lines for the corresponding three linear regression models. For each plot, the two sided t-test on the regression slope and the associated R-squared are also given.

as part of a hierarchical model, but the conceptual difficulty with that approach is that the likelihood function would have to account for spatial dependence between the annual maxima observed at different stations. The assumption stated above, although it might appear unreasonable at first thought, is certainly more reasonable than attempting to model the spatial dependence between the annual maxima, especially when such dependence is of no particular interest.

That said, the variables `mu.x` and `sigma.x` may be used to declare a prior distribution for the GEV parameters of station 01AJ007 i.e.

```
> prior.x <- function(p)
+   dmvnorm(x=c(p[1:2],log(p[3])),mean=mu.x,sigma=sigma.x)
```

Finally, the model may now be estimated using `metropolis` - note the difference between the arguments specified in previous occurrences of `metropolis` and the one below.

```
> inits.1 <- gev.inits(model.x[['x']])
> inits.1[2] <- log(inits.1[2])
> bayes.x.1 <- metropolis(model=model.x,iter=5000,
+   trans.list=trans.list.x,start=inits.1,variance=diag(.1,3),
+   prior=prior.x,pass.down.to.C=TRUE)
> plot(bayes.x.1)
```

Inspection of the trace plots shown by Figure 12 suggests that the Metropolis algorithm could use some tuning in order to improve mixing of the chain. Using the guidelines given by Gamerman & Lopes (2006) and Carlin & Louis (2009), for example, the following code chunk illustrates how tuning may be carried out using the output from `bayes.x.1`.

```
> start.x <- bayes.x.1[['M']]
> variance.x <- bayes.x.1[['V']]
> bayes.x.2 <- metropolis(model=model.x,iter=15000,
+   trans.list=trans.list.x,start=start.x,variance=variance.x,
+   prior=prior.x,pass.down.to.C=TRUE)
> plot(bayes.x.2)
```

Figure 13 suggests that the posterior distribution may be well approximated by the correlated draws contained in `bayes.x.2`, such that it now seems reasonable to use the latter to approximate the posterior distributions of some quantiles of interest, which are shown by Figure 14. Although somewhat sparse, the posterior distribution of the 100-year flood is still very informative, especially given the relatively small sample of size 7 used in its estimation.

```
> prior.draws <- rmvnorm(100000,mu.x,sigma.x)
> prior.draws[,2:3] <- exp(prior.draws[,2:3])
> p <- c(.5,.9,.99)
> Q.p.post <- sapply(as.list(p),function(h)
+   distr(x=h,dist='gev',param=bayes.x.2[['sims']],type='q'))
> Q.p.prior <- sapply(as.list(p),function(h)
+   distr(x=h,dist='gev',param=prior.draws,type='q'))
> layout(matrix(1:length(p),nr=1))
```

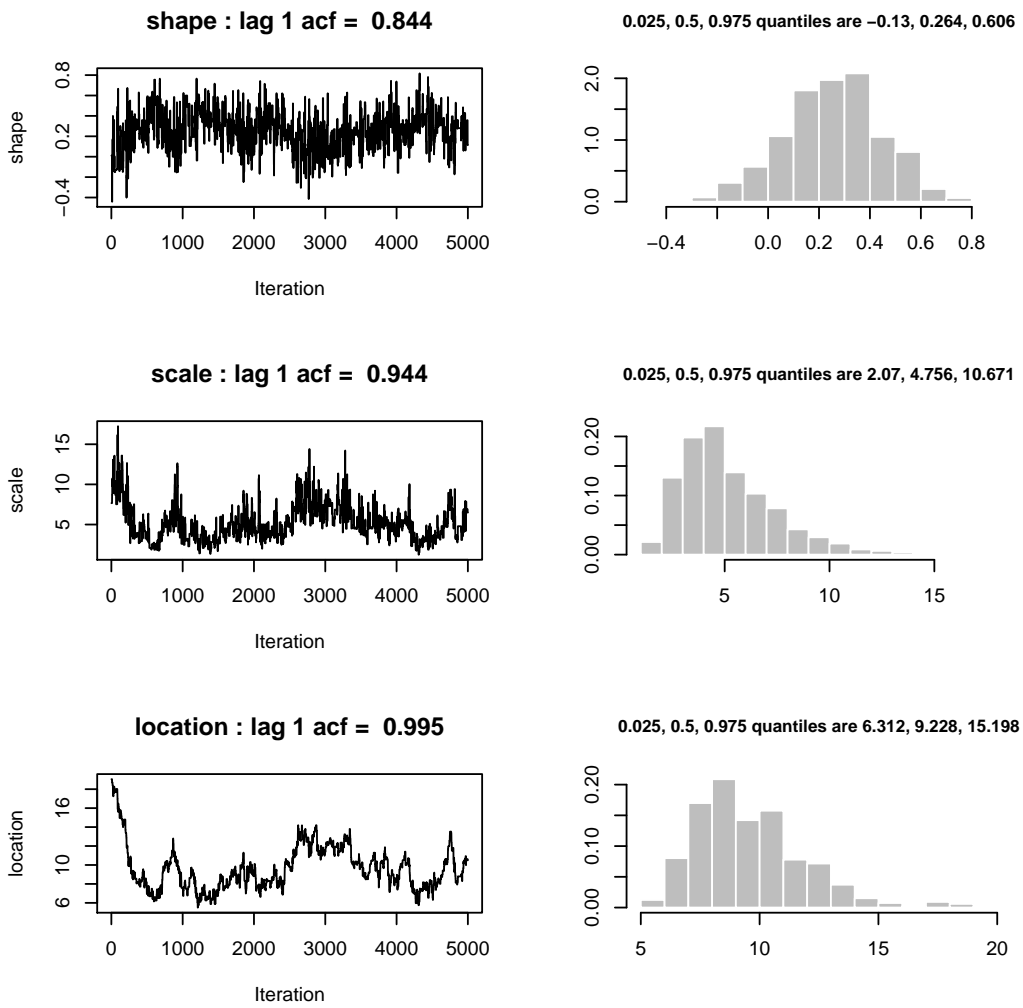


Figure 12: Trace plots for the object `bayes.x.1`.

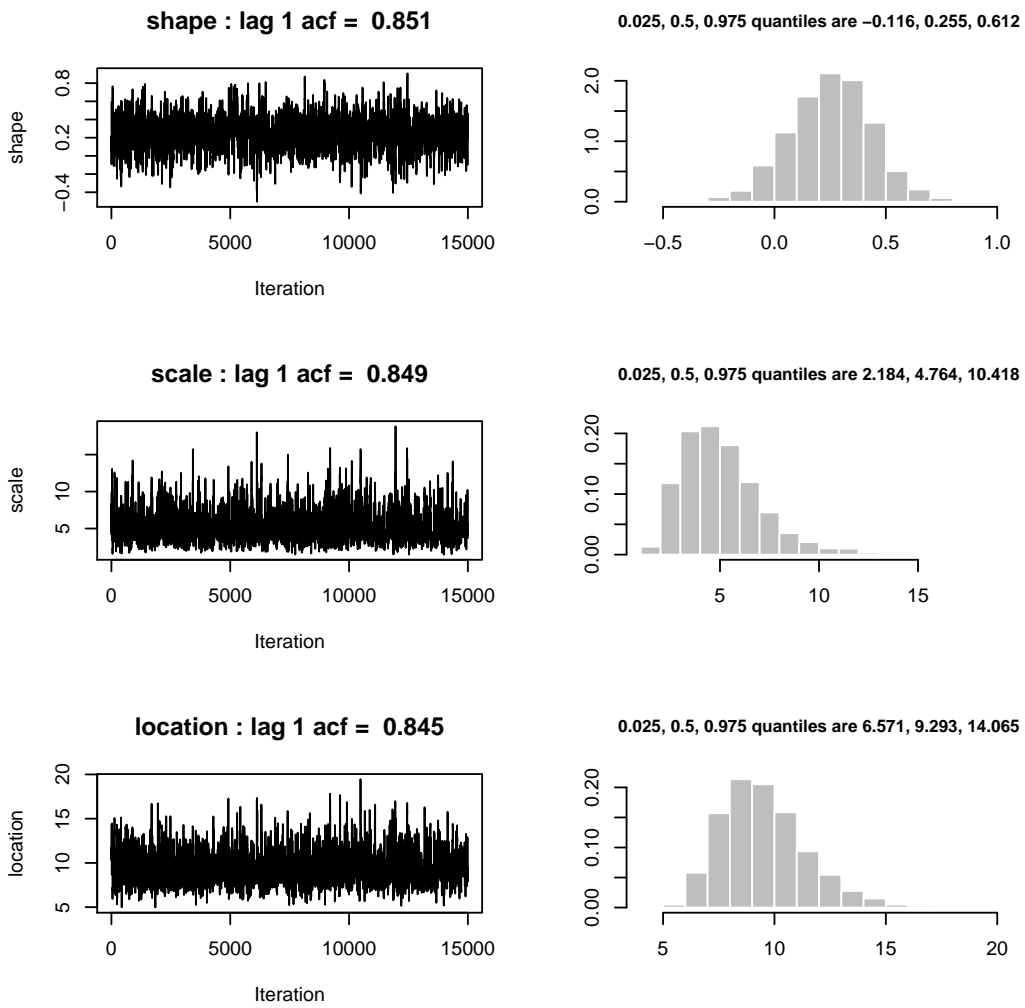


Figure 13: Trace plots for the object `bayes.x.2`

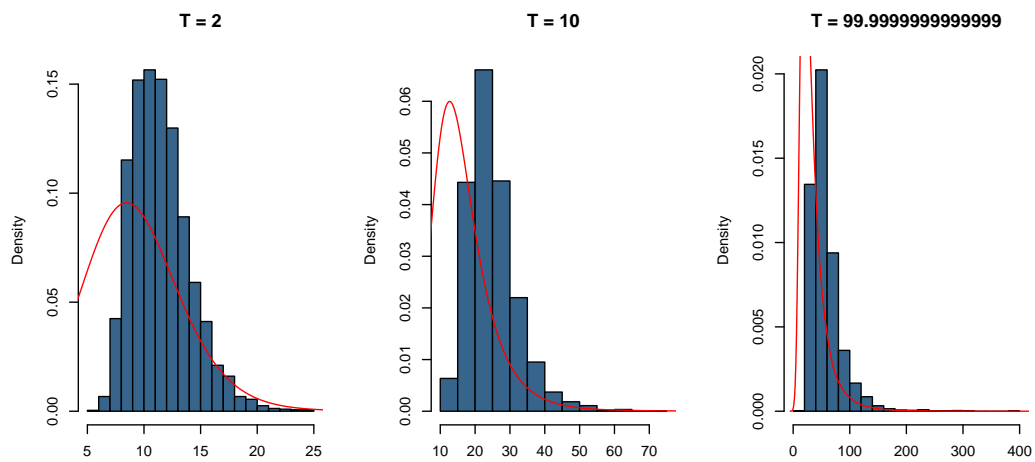


Figure 14: Posterior quantile distributions (blue histograms) obtained using the `bayes.x.2` object. In addition, the density estimates of the corresponding prior distributions (red curves) are also presented.

```

> for(i in 1:ncol(Q.p.post)) {
+   hist(Q.p.post[,i],freq=FALSE,col='steelblue4',
+       main=paste('T = ',1/(1-p[i]),sep=''),xlab='')
+   lines(density(Q.p.prior[,i],bw=3),col='red')
+ }

```

References

- Aucoin, F. 2011. *FAdist: Distributions that are Sometimes used in Hydrology*. R package version 2.0; <http://cran.r-project.org/package=FAdist>.
- Box, G.E.P., & Tiao, G.C. 1973. *Bayesian Inference in Statistical Analysis*. Addison-Wesley Publishing Company.
- Carlin, B.P., & Louis, T.A. 2009. *Bayesian Methods for Data Analysis*. Chapman & Hall.
- Coles, S. 2001. *An Introduction to Statistical Modeling of Extreme Values*. Springer - Verlag London.
- Cox, D.R., & Hinkley, D.V. 1979. *Theoretical Statistics*. Chapman and Hall.
- Davison, A.C., & Hinkley, D.V. 1997. *Bootstrap methods and their application*. Cambridge University Press.
- Gamerman, D., & Lopes, H.F. 2006. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Chapman & HALL/CRC.
- Gelman, A., Carlin, J.B., Stern, H.S., & Rubin, D.B. 2004. *Bayesian Data Analysis*. 2 edn. Chapman & Hall.
- Gilks, W.R., Richardson, S., & Spiegelhalter, D.J. 1996. *Markov Chain Monte Carlo in Practice*. Chapman & Hall.
- Hogg, R.V., Craig, A., & McKean, J.W. 2004. *Introduction to mathematical statistics*. Prentice Hall.
- Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., & Teller, E. 1953. Equation of State Calculations by Fast Computing Machine. *Journal of chemical physics*, **21**, 1087–1091.
- Rice, J.A. 2003. *Mathematical statistics and data analysis*. Duxbury.